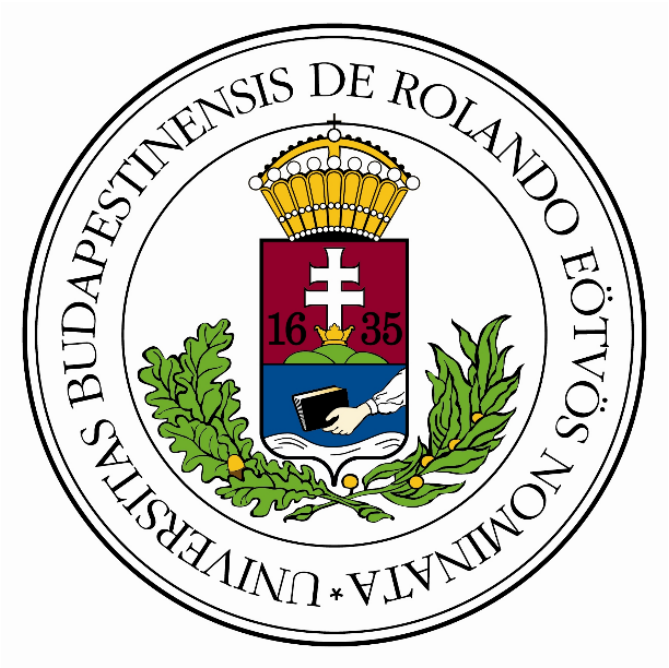


Az informatika (programozás) oktatásának módszertani kérdései

Doktori értekezés

Szalayné Tahy Zsuzsanna

2021

EÖTVÖS LÓRÁND TUDOMÁNYEGYETEM INFORMATIKA DOKTORI ISKOLA
INFORMATIKA SZAKMÓDSZERTAN DOKTORI PROGRAM

AZ INFORMATIKA (PROGRAMOZÁS) OKTATÁSÁNAK MÓDSZERTANI KÉRDÉSEI

SZALAYNÉ TAHY ZSUZSANNA

ISKOLAVEZETŐ: DR. CSUHAJ VARJÚ ERZSÉBET
AZ MTA DOKTORA, TANSZÉKVEZETŐ EGYETEMI TANÁR

PROGRAMVEZETŐ ÉS TÉMAVEZETŐ: DR. ZSAKÓ LÁSZLÓ
TANSZÉKVEZETŐ EGYETEMI DOCENS
ELTE IK MÉDIA- ÉS OKTATÁSINFORMATIKAI TANSZÉK

DOI: 10.15476/ELTE.2021.188

BUDAPEST, 2021

Tartalom

Köszönet

Mindenekelőtt, szeretném megköszönni mindazok segítségét, támogatását, akik e disszertáció létrejöttében akarva, akaratlanul közreműködtek.

Először is, köszönöm családomnak, legfőképp férjemnek, hogy eltűrték maximalista és munkamániás hóbortjaim és gondoskodtak arról, hogy mindig legyen kéznél tej.

Nagyon régről tartozom egy köszönettel a Nyílt Társadalomért Alapítványnak a Számítógépes iskola támogatásért, amellyel elindított az innovatív informatikatanár pályán.

Köszönöm témavezetőmnek, Dr. Zsakó Lászlónak, hogy informatikatanári énemet év(tized)eken át nevelgette, fejlesztette, majd, amikor a doktori iskola elvégzésére elszántam magam, a kutatási elképzeléseimet támogatta, úgy vezetett a disszertáció megírása felé, hogy közben megőrizhettem önállóságomat egy multidiszciplináris terület kutatásában.

Köszönöm Dr. Szlávi Péternek, az ELTE IK Médiainformatika Tanszék munkaközösségének és hallgatótársaimnak, hogy a programtervező informatikusok oktatását megismerhettem, gondolataikba, módszereikbe beavattak.

Köszönettel tartozom az Ady Endre Gimnázium, a Szent István Gimnázium és a Veres Péter Gimnázium tantestületének, az informatikatanárok és rendszergazdák szakmai szervezeteinek, munkacsoportjainak a rengeteg tapasztalatért, véleményért, amely munkámat kíséri, és kutatásom alapjául szolgál immár több, mint 30 éve.

Kiemelten köszönöm Dr. Tevesz Gábor oktatási dékánhelyettes támogatását a Budapesti Műszaki és Gazdaságtudományi Egyetem Villamosmérnöki és Informatikai Kar (BME VIK) mérnökinformatikus képzésének megismerésében, különös tekintettel arra, hogy több szempontból „más világból” származó lényként elég sok gondot okoztam.

Köszönöm Dr. Gajdos Sándornak, Dr. Szeberényi Imrének, Dr. Selényi Endrének, Dr. Pataricza Andrásnak, Dr. Recski Andrásnak, Dr. Orosz Lászlónak és még számos oktatónak – akiknek az óráit látogathattam, módszereit megfigyelhettem – a türelmét, őszinteségét.

Köszönöm az együttműködést a BME VIK mérnökinformatikus képzés kidolgozásában résztvevő oktatóknak, demonstrátoroknak. Közülük többen a kutatás során váltak hallgatóból mérnökké, kutatóvá, egyetemi munkatárssá, többen ösztöndíjasként részt vettek a középiskolai tehetséggondozásban, vagy mentorálták volt diákjaimat, így a tantárgyfejlesztés módszertani kérdésein túl saját pályájuk és diákjaim pályájának követésére is lehetőségem volt, ami rengeteg

tapasztalatot adott. Munkájukkal, együttműködésükkel a kutatásom nagyon nagy látószögű lehetett. Név szerint szeretném kiemelni Dr. Vörös András, Dr. Szárnyas Gábor, Dr. Micskei Zoltán, Horányi Gergő, Búr Márton, Dr. Molnár Vince, Estók Dániel, Dobra Gábor nevét, akikkel együttműködésem években mérhető.

Sem a matematikában, sem a mérnöki világban nem szokás megköszönni a kutatás tárgyának, hogy rendelkezésre állt. De ez a kutatás az oktatásról szól, a diákok, hallgatók képzésének módjáról, lehetőségeiről. Köszönöm diákjaimnak és hallgatóimnak a közös munkát. Dr. Darvas Dánielnek egy feladatgyűjteménybe azt írtam, lehet, hogy sokat tanult tőlem, de én talán még többet tőle. És tanultam minden diáktól. Az ilyentől ezt, az olyantól azt. Aki sikeres volt, attól a siker titkát, aki szenvedett attól a szenvedésének okát, aki bukott, attól a talpra állás módját. Köszönöm a partnerséget a tanulásban.

Dr. Czirkos Zoltán szerepét a kutatásomban jellemzi kilenc közös publikációnk. Megtanított a mérnöki szemléletmódra, tanultam tőle C és C++ nyelven „mérnökinformatikusi programozni”. Engedte oktatásának módszertani elemzését, emellett partner volt oktatásmódszertani kísérletekben, felmérésekben. Éveken keresztül elviselte teljesen más világképemet, Y generációs egyetemi docensként az X generációs informatikatanárt. Néha szóban, de főleg e-mailben zaklattam, négy év alatt több, mint 300 threadben… Köszönöm.

1 Bevezetés

1.1 Gyökerek

A diploma megszerzése után 25 évvel, az informatikatanári tevékenységem 20 éve után, 2013-ban kezdtem meg doktori képzésemet. Már az informatikatanári diploma megszerzésének is az volt a motivációja, hogy jobban tudjak tanítani. A „Számítógépes iskola a nyílt társadalomért” című pályázat [0] keretében elnyert támogatás – a diplomám mellett – hatalmas lökést adott az innovációhoz, az informatika (akkor még számítástechnika) tantárgy oktatásának és az IKT más tantárgyon belüli felhasználásának a fejlesztéséhez. Több doktori kutatásban (például [2]) vehettem részt mint megfigyelt személy, később az oktatási stratégiák kidolgozásában, képzési tematikák fejlesztésében, érettségi fejlesztésében [3], tankönyvek, feladatgyűjtemények írásában is részt vehettem. A kihívások folyamatos változása informatikai és pedagógiai továbbképzésekre ösztönzött, így minőségbiztosítási és tehetséggondozási képzéseken is részt vettem, majd szakvizsgát is tettem. Tehetséges diákjaim a versenyeken, az érettségin és az egyetemen is jól teljesítettek. Érzékenyen érint, amikor ismereteimnek ellentmondóan, tudományos, illetve pillanatnyi politikai potentátok kijelentik, hogy a programozás egyesek számára nem megtanulható – cáfolat: [4] –, illetve a mai gyerekek már úgy születnek, hogy tudják használni a kütyüket [5] – cáfolat: [6]. A 2013-as kerettanterv egyik érve, hogy az informatikát minden tanár – az informatikai analfabéta is – tudja tanítani. Ennek ellenében, az „Ensuring Exemplary Teaching in an Essential Discipline:…” [7] tanulmányban az informatikatanár szakmaiságának követelménye: a tudományterületbéli szaktudás, szakmai tudás pedagógiából és jártasság az oktatásszervezési kérdésekben. A kutatásom során ezeket a területeket és egymásra hatásukat elemeztem, mélyebben megismertem az informatika tantárgyi követelményeket, ezen belül a programozás szakmai hátterét, a felmerülő problémákra és ellentmondásokra megoldásokat kerestem.

A kutatás részeként hazai és nemzetközi szakmódszertani konferenciákon vettem részt és olvastam a kapcsolódó tudományos szakirodalmat, valamint figyelemmel kísértem az állami és civil kezdeményezéseket is. A közoktatási informatika képzés szakmai tartalmának meghatározásához a továbbtanulás szempontjából meghatározó, két legnagyobb magyarországi informatikusképző intézmény – az ELTE IK, illetve a BME VIK – képzéseit alaposabban tanulmányoztam. Számos előadáson, gyakorlaton és laboron hospitáltam. Ahol lehetett, vendéghallga-

tóként kipróbáltam a tanulást, demonstrátorként, vendég óraadóként, korrepetitorként az oktatást. Részt vettem tananyag lektorálásában, dolgozatok javításában, a hallgatók haladásának figyelésében, az eredmények kiértékelésében.

Saját bőrömön, mindennapi tevékenységem során tapasztaltam meg a programtervező informatikus és a mérnökinformatikus szemléletmódja közötti különbséget, gyakran ellentétes nézőpontot. Tanítottam úgy, hogy pedagógiai módszerekkel pótoltam szakmai hiányosságaimat, és láttam, hogy pedagógiai ismeretek nélkül milyen problémákkal küzdenek az oktatók. Mentoráltam ösztöndíjprogram keretében szakkört tartó egyetemistát, együtt dolgoztam táborban szakmai segítséget nyújtó hallgatókkal, demonstrátorokkal, gyerekek oktatását vállaló szülőkkel és vállalati alkalmazottakkal, akik oktatást vállaltak szabadidejükben. A hallgatók tömeges felmérése [146, 147, 148] mellett igyekeztem diákok és hallgatók haladását, további pályáját nyomon követni. Több diákommal, akiket középiskolában éveken át tanítottam, találkoztam az egyetemen kutatásom során. Egyes diákjaim pályamódosítás során kerültek levelezős hallgatói csoportomba. Kuriózum, hogy volt, akivel együtt látogattam órát a BME mérnökinformatikus képzés 1. évfolyamán, utána felkésztőként segítettem az emelt szintű informatika érettségi teljesítését, ezt követően tanítottam neki a programozást az ELTE-n levelező tagozaton. Ezen egyedi esetek összekötik a csoportokon kipróbált vagy megfigyelt tapasztalatokat.

Kutatásom során rengeteget tanultam programozásból, tervezői és mérnöki gondolkodásmódból. Ezeket konkrétan fel tudom használni tanári munkám során. A számítástechnikatanári diplomám megszerzésekor nagyjából tudtam, hogy mit kell majd tanítanom. Nagyjából azt is tudtam, hogy folyamatosan változni fog a tananyag, mert fejlődik a világ, ezen belül robbanás szerű változást élünk meg az informatika területén. Tudtam, hogy folyamatosan kell tanulnom. Módszertanból kaptam ötleteket, de hogy mit, mikor, hogyan alkalmazzak, arra magamtól kellett rájönnöm. Folyamatosan új, jobb megoldásokat kerestem, a lehetőségek és helyzetek függvényében. Az elmúlt években rendszerezetté váltak az ösztönös megoldások: tudatosan kombinálom a szakmódszertani és pedagógiai szempontokat egy-egy helyzet elemzése és megoldások keresése során, amit a 8. fejezetben, illetve több cikkben [121, 127, 133, 137, 159, 160] mutatok be. A fentebb említett tanulmányban [7] a szükséges ismeretek felsorolásként, elemenként olvashatók, disszertációm témája nagyon hasonló, de más szempontú: az egyes szakmai területek egymást erősítő szakmódszertani-pedagógiai rendszerét mutatom be.

1.2 Kutatási módszerek

Az ELTE IK Informatika Doktori Iskolában végeztem doktori tanulmányaimat, de a téma nemcsak az ELTE IK kutatási körébe tartozó informatikatudományhoz kapcsolódik. A mérnökinformatika erős utánpótlásigénye miatt a mérnöki tudományokat is vizsgáltam. Egy informatikatudományi disszertáció gyakran tartalmazza matematikai modellek igazolását, levezetését, bizonyítását. Egy mérnöki disszertációban nagy mennyiségben kellenek mérési adatok, melyek a mérnöki megoldást alátámasztják. Kutatásom során igyekeztem mindkét kutatásmódszertant alkalmazni, de a kutatás fő területe az oktatás, az informatika oktatása. Oktatás-neveléssel a neveléstudomány foglalkozik, középpontjában az ember, az ember megváltoztatása, tanítása áll. A disszertációm az oktatás módszertani kérdéseiről szól, azaz arról, hogy hogyan tanítható az ember informatikára. Ezért kutatásom jelentős részében a neveléstudományra jellemző kutatási módszereket használtam. Disszertációm a kutatási téma és módszerek tekinteténen hasonló Päivi Kinnunen, „Challenges of Teaching and Studying Programming at a University of Technology…" [8] disszertációjához, de az oktatás tartalmát is vizsgáltam, a finn helyett magyar viszonyok között. A disszertációm nem 200 oldal, hanem csak 120 oldalas, amit nem 70 hanem 200 oldal referencia és melléklet egészít ki.

A kutatásnak minden részterületén végeztem analitikus kutatást, amelynek célja a tudományos elméleti ismeretek és a gyakorlatban alkalmazott szabályzók elemzése és rendszerezése. Emellett a kutatás során igyekeztem olyan empirikus kvalitatív módszereket is alkalmazni, amelyek nem módosítják a megfigyelés tárgyát, illetve a módosulás közvetlenül detektálható.

Megfigyeltem a környezetemet – a diákjaimat, a kollégáimat, a hallgatókat, az ismerősöket – sok esetben úgy, hogy ők a megfigyelésről nem tudtak. Egy-egy elejtett szó, élethelyzet, természetes reakció bekerült az „esetek gyűjteményébe". Az interjúkat, amennyire lehetett, kötetlen beszélgetések, levelezések formájában, a tanítási folyamat részeként készítettem. Kérdőíves felmérés helyett is inkább konkrét helyzetekről alkotott véleményeket elemeztem, problémák megoldási módjait figyeltem meg. Sok esetben saját magamat figyeltem meg, elemeztem reakcióimat, teljesítményemet mind a tanítás, mind a tanulás területén.

2 Tézisek

A kutatás során szembesültem azzal, hogy nemcsak módszertani kérdéseket kell megvizsgálnom, hanem az oktatás tárgyát is. Tisztáznom kell, mit jelent az „informatika”, mit jelent a „programozás”; mit nevezünk informatika tudománynak. Az informatikatudomány nagyon fiatal és nagyon dinamikusan fejlődik. (Nem szeretném azt mondani, hogy születőben levő, mert azt gondolom, hogy már létezik.) Meg kellett határoznom, hogy az informatikus szakmák folytonos változása mellett mi az az állandó szakmai tudás, amit az informatika tantárgy megalapoz; ebből következően mit jelent informatikát tanítani. A középiskolai tanítás szempontjából – így kutatásom során – kérdés, hogy elkülönül-e szemléletmódjában, tartalmában az informatika a matematikától; a különböző informatikus továbbtanulási irányokból lecsapódó, néha egymásnak ellentmondónak látszó paradigmák oktathatók-e egységes szemléletmóddal és ha igen, ez hogyan valósítható meg. A programtervező informatikus és a mérnökinformatikus képzést egymással és a többi tantárgyban tanított ismeretekkel összehasonlítva, a képzések eredményességének tanulmányozása során tapasztaltakat fogalmaztam meg az első tézisben (T1):

Tézis: Az informatikatudomány gondolkodási módszereiben és eszközeiben egységes rendszert alkot, amelynek sikeres oktatásához tudományspecifikus oktatásmódszertan szükséges.

A közoktatásban az informatika oktatásának nem egyetlen célja az informatikus képzésre felkészítés. Az informatikaoktatást alapvetően meghatározza, hogy mit tartalmaz a tanterv, a megvalósítás során jellemző a szakadék az informatikai eszközök használata és az alkalmazói szoftverek oktatása, valamint az informatikus szakmák megalapozásaként a programozás oktatása között. Az elsőről él az „ez csak játék” közvélekedés, a másodikat sokan tartják megtanulhatatlannak. E kettéválasztás megszüntetésén szinte pályám kezdete óta dolgozom, ennek eredménye a második tézis (T2):

Tézis: Minden, informatikatantervben előírt témát informatikatudományi megközelítéssel oktatva együtt fejleszthető az informatikai gondolkodás, az alkalmazói készségek és a programozási készségek.

Másképp: az informatikát úgy kell oktatni, hogy az alkalmazások oktatása egyben programozásoktatás is legyen; a programozás oktatása fejlessze az alkalmazói szoftverek célszerű és hatékony felhasználását a problémamegoldásban, ezzel bármelyik oldalt tekintve, az informatikai gondolkodást fejlessze.

Az első tézisben (T1) megfogalmazott „sikeres oktatás”, a második tézisben (T2) megjelenő „oktatva együtt fejleszthető” további kérdéseket vet fel. Az oktatást akkor nevezem sikeresnek, ha a tanuló elsajátítja a tudást, megszerzi a megfelelő képességet, készséget. Az oktatás során a fejlesztés egy folyamatot jelez.

Az oktatás sikerességének elválaszthatatlan része a tanuláson keresztül szerzett tudás és képesség jellemzése, amire egy, az oktatás során használható, folyamatot is leíró modellt készítettem. Ez a (Learning Activity Unit) LAU-modell. Kérdés, hogy a modell összhangban van-e más, az oktatásmódszertanban használt modellekkel, illetve alkalmas-e a szignifikáns jellemzők meghatározására egy adott tudás állapotának, illetve tanulás folyamatának tekintetében (T3).

Tézis: A LAU alapú leírás, valamint ezek kombinációjaként a LAU-modell egy eszköz a tudás-, a készség- és a képességelemek, illetve a tanulási és tanítási folyamat leírására, jellemzésére.

Az oktatás hatékonyságát – és ezzel minőségét – a tanulás eredményén keresztül vizsgálhatjuk: a potenciális tanulók mekkora aránya, milyen tudás- illetve készségszintre jut el. Az egyik legfontosabb oktatásmódszertani kérdés, hogy mi okozza a sikertelenséget. Van-e olyan módszer, amivel mindenkit meg lehet tanítani programozni? Úgy tűnik, egy ilyen módszer nincs, a módszertani sokszínűség, adaptivitás vezethet a megoldás felé. Kérdés, hogy milyen szempontok, tulajdonságok alapján lehet az adott helyzetben (az adott tanulóhoz) megfelelő módszert megtalálni, illetve – határesetként – van-e taníthatatlan tanuló, akit fel kellene menteni az informatikaoktatás alól, a programozás tanulása alól, akinek az informatikai gondolkodás képessége nem fejleszthető. A készség fejleszthetőségéről szól a negyedik tézis (T4).

Tézis: Az informatikai gondolkodás – ezzel együtt a programozás – képességének a fejlesztését és gyakorlását a motiváció, az érzelmek, a mentális állapot katalizátorként segíti vagy blokkolja.

A tanítási tapasztalat azt mutatja, hogy a negatív prekoncepciók és mentális gátak legyőzését követően az informatikai gondolkodás és a programozási alapkészségek aktiválhatók, fejleszthetők. Másik oldalról, személyre szabott, motiváló helyzetben az informatikai gondolkodás pozitív élményt jelent, ami fejlesztésének motorja is [9].

A fenti kérdések és a tézisek az informatikus szakmáról, az informatikaoktatás tartalmáról, az oktatásmódszertanról és a pedagógiai gyakorlatról szólnak. Négy terület, melyeknek belső összefüggéseit és egymásra hatását is vizsgáltam.

3 A tanulás és tanítás

Az egyetemi óralátogatások, megbeszélések során tapasztaltam, hogy az oktatók a tananyag elmondását tartják feladatuknak, amit a hallgatók meghallgatva jegyzetelnek, rögtön meg is értenek, tehát már alkalmazni is tudják. Néha az volt az érzésem, hogy a tanulás sokak számára azt jelenti, hogy a hallgató a „fejébe betölti a tudást", avagy – informatikus szóhasználattal – „elmenti a memóriájába" a tananyagot. Középiskolai tanárokkal, szülőkkel beszélgetve, tanárjelöltek óráit látogatva lényegében ennek a modellnek számos megnyilvánulását tapasztaltam: „Figyelj, mert csak egyszer mondom el!", „Ha nem mondom el én a tananyagot, akkor nem fogja megérteni!", „Hogy képzeli a tanárnő, hogy nem írja fel a táblára a megoldást és nem magyarázza el, hanem a gyerekemnek kell a könyvből kikeresnie?", „Hogy hogy nem tudja alkalmazni a formulát a hallgató, hiszen tavaly szerepelt az előadáson, benne van a jegyzetben. És ötöst kapott."

A fentiekhez hasonló kijelentések, helyzetek hatására írtam le, hogy hogyan modellezem a folyamatot. Ismereteim bázisa az 1998-ban kiadott, Falus Iván szerkesztette Didaktika [11] tankönyv, de a leírás során fontos volt, hogy informatikus számára könnyen érthető, a mindennapi munkában használható, vizualizálható (infografikusan megjeleníthető) legyen a modellem. Így született a Learning Activity Unit (LAU), illetve az ezek rendszeréből képezhető LAU-Model. Az elképzelésemet helyi viszonyoknak megfelelően finomítottuk és használtuk a *Programozás alapjai 1*. tantárgy tananyagának elemzéséhez, fejlesztéséhez.

3.1 Learning Activity Unit – LAU

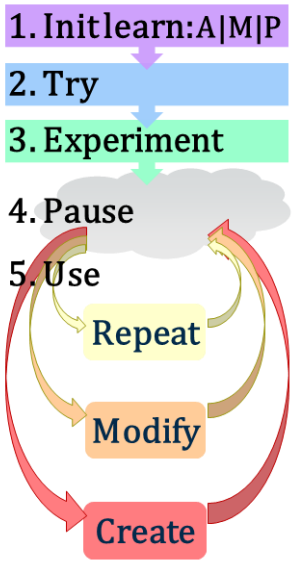
A Learning Activity Unit egy tanulási sablon, elemző eszköz, ami egyaránt használható tantervek, tanmenetek minősítésére, tanítási és tanulási célok megfogalmazására, a tanulási folyamat során felmerülő problémák detektálására, illetve a kimenet (az elvárások) pontos meghatározására. Az az egység, aminek megtanulási folyamatát jellemezzük, lehet akár egy fogalom, definíció, egy eljárás, egy koncepció, egy témakör, egy megtanulandó tevékenység. Az egység megtanulásához szükség lehet előismeretre, amely az előismeretek egységeinek megnevezésével és az adott egységhez kapcsolódásuk módjával írhatók le.

A LAU és LAU-modell használata lehetővé teszi a tanulással és a tudással kapcsolatos állítások gyors és pontos megfogalmazását, amellyel a tanítási helyzetekre jobban fel lehet készülni; „futási időben", azaz a tanórán, a helyzethez alkalmazkodó döntést lehet hozni. Magasabb szinten a LAU és LAU-modell alkalmas eszköz tantárgyfejlesztéshez, tematikák elemzéséhez, mérés-értékelési módszerek kidolgozásához, minőségbiztosítási elemzésekhez.

3.1.1 Learning Activity Unit fázisai

A LAU leírása és ábrázolása – igazodva az informatikában használatos kifejezési módhoz – az algoritmizálás, illetve az állapotgép leírásának elemeit tartalmazza. Az ismeret modellekre [11, 12, 15, 16, 17, 18, 19, 20] jellemző „piramis" egymásra épülő szintjei helyett az átmeneteket és időbeliséget hangsúlyozza (1. ábra).

1. **Initial learn**: a tanulás folyamatában az első lépés a tananyag megismerése. Ennek sokféle módja lehet, de a folyamat leírásához, a megismerés minőségére három jellemzőt definiálunk:
   A) **Active** megismerés: az új ismeret és a korábbi ismeretek között azonnal kialakulnak kapcsolatok. Az ismeretnek előkészített helye van, az új ismeret azonnal használható.
   M) **Moderated** megismerés: a tanuló tudja, hogy hol lesz szüksége az ismeretre; jegyzet segítségével vagy más ismeretekkel kapcsolva képes felidézni a tanultakat.
   P) **Passive** megismerés: a tanuló nem képes összefüggéseiben megérteni, megjegyezni a tudáselemet. Az új ismeretelemek nem kerülnek kontextusba más ismeretekkel (céltalan memorizálás, megjegyzés későbbre halasztott megértéssel, nem tudatos észlelés).
2. **Try**: az ismeret első kipróbálása, alkalmazása adott kontextusban; megértés ellenőrzése.
3. **Experiment**: az ismeret részleteinek kifejtése; kapcsolatok, összefüggések megértése; felhasználás gyakorlása.
4. **Pause**: szünet – más ismeretekkel foglalkozás (szűrés, felejtés).
5. **Use**:
   a. **Repeat** az ismeret felhasználása a tanult formában, a tanultak **ismétlése**.
   b. **Modify**: az ismeret felhasználása **módosít**ott, de a tanulthoz hasonló formában környezetben; tipikus helyzetekben.
   c. **Create** az ismeret szükség szerinti, célorientált felhasználása, nem várt helyzetben, **alkotó**, kreatív módon.



1. ábra: A LAU fázisai

← Újra a 4. pont, vagy – hosszú szünet után, illetve más környezetben, más megközelítéssel – újra az 1. ponttól.

Egy LAU időben több éven át tarthat. Például, ha egy programozási nyelvet valaki megtanul és használja nap mint nap, akkor egy 5c fázisú tudásról beszélhetünk. Azonban, ha ezután évekig nem használja az adott nyelvet, akkor ez a tudás erősen elkopik, idővel 5a szintre csökken. Sőt, az is előfordulhat, hogy célszerűbb újratanulnia, ami valószínűleg egy 1A jellegű fázis lesz.

Egy LAU 5c fázisa alapja lehet egy másik LAU 1. fázisának, ha az például extrapolációként vagy asszociációként épül a korábbi ismeretekre.

Egy nagyobb vagy összetett tananyagban az egyes LAU-ok egymáshoz való viszonyának megadásával modellezhetjük a tananyag felépítését. Ezt nevezem a tananyag „LAU-modell"-jének. A modellben egy LAU előfeltétele más LAU-ok megnevezésével és fázisainak megadásával írható le. A tanítási folyamatban egyes LAU-ok egymásra épülése, párhuzamos végzése vagy „közbevetett", beépülő megvalósulása is lehetséges. A modell alkalmas eszköz tantervek és tanmenetek Pólya-féle problémamegoldási megközelítéssel [32] történő készítéséhez és a megvalósíthatóságuk vizsgálatára.

A LAU a tanulási folyamat elnagyolt, egyszerűsített, ezzel együtt pontatlan, vázlatos leírása. A témával kapcsolatos tudományos kutatások eredményeit intuitíven kombinálja. Célja az oktatásmódszertani és tanulásmódszertani elméletek szintézise a gyakorlat szintjén, a tanulásról szóló megállapítások közös „nyelvének" megteremtése.

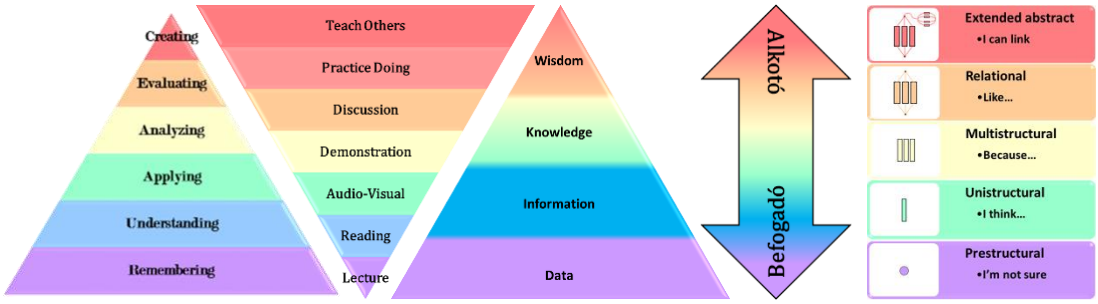
A LAU egyes fázisainak ($^{L}1 \ldots {}^{L}5$)[1] részletesebb leírása a 0. mellékletben olvasható, a LAU-ok kapcsolódását is figyelembe vevő LAU-modell formalizált leírása a II. mellékletben található. A LAU-t a kutatás során, publikációkban [77, 146, 147] és napi szinten a tanítási gyakorlatban is használom.

3.1.2 Pedagógiai rendszerek megvalósulása a LAU-ban

Kutatásom egyik alappillére az informatikaoktatás módszertanának specifikálása, azaz a pedagógia és neveléstudomány területéről az informatikaoktatás szempontjából releváns ismeretek analízise, a LAU-modellel való összevetése. A III. mellékletben olvasható néhány taxonómia jellemzése, összehasonlítása. Scott Smith elemzésének [21] kibővítésével megállapítható, hogy a Bloom [11], a SOLO [13], a DIKW [16] taxonómia és a tanulást jellemző Learning Pyramid [19] gyakorlati alkalmazása közös struktúrába szervezhető (2. ábra), ezt a struktúrát követi a LAU is. Emellett azonban a felsorolt taxonómiák csak állapotokat definiálnak, célként a fejlődést jelölik meg. A gyakorlatban tapasztalt tudásbéli stagnálást, illetve visszalépést a fellelt modellek közül csak a PRIMM-modell [22] értelmezi. A LAU modellezésben ezen túlmenően is fontos szerepe van az időnek. A túl gyors, túl nagy mennyiségű tananyag elsajátításában akadály, hogy nincs idő a feldolgozásra, ugyanakkor a túl ritkán használt ismeret elfelejtődik. Ezért a LAU integrálja a tanulásra jellemző időbeliséget is: a megtanulás fázisait, illetve felejtés-ismétlés ciklust. Ebben Ebbinghaus felejtési görbéjének [24] jellemzői jelennek meg, az

[1] A LAU formalizálásában az egyes fázisokat a ^{L} jellel különböztetem meg egyéb számozásoktól és jelölésektől.

ismeret felhasználási módjának változása összhangban van Sweller Cognitive Load Theory megállapításaival [25].



2. ábra: Bloom Learning Pyramid DIKW Kiegészítés SOLO

3.2 A tanulási folyamatot befolyásoló tényezők

A LAU és LAU-modell a tanulás/tanítás tartalmának, egymásra épülésének leíráshoz, az aktuális állapot jellemzéséhez, a célok és eredmények megfogalmazásához, anomáliák kimutatásához adnak keretet. A LAU és LAU-modell használata az (informatika) szakmódszertanból [10] elsősorban a didaktika, másodsorban a szakpedagógia leírását teszik lehetővé. Nem szólnak arról, hogy a tanulás/tanítás hogyan történik, ehhez további szakmetodikai és szak-filozófiai megközelítési módokat kell tekintetbe venni.

3.2.1 Folyamatok leírása LAU-ban és LAU-modellben

A LAU a tervezett tanulást írja le. A megvalósulás során előfordul, hogy az első két fázis megcserélődik: Előbb tapasztalunk meg valamit és csak ezt követően, az élmény hatására tájékozódunk arról, hogy mi volt a tapasztalat. A megtapasztalást követő megtanulást – ha a tapasztalás nem életveszélyes, fájdalmas vagy „keserű” – tervezni is lehet, külön előnye, hogy az Initial learning fázis ebben az esetben jellemzően Active (1A). A jelenség részletesebb leírása az IV/1. mellékletben olvasható. Metodikai oldalát tekintve, ez a felépítés egyéni és kiscsoportos oktatás esetén valósítható meg igazán. Tömeges, nagy csoportok oktatásánál a tanulók sokfélesége miatt eltérő hatásokat tapasztaltam: Van, akinek korábbról ismerete vagy tapasztalata van; eseti, hogy felcserélődik-e a $^{L}2$ és a $^{L}1$ fázis, esetleg sokkal előrébb jár. Van, akinek – a szándék ellenére – nem pozitív élmény a megtapasztalás, ami a további tanulás elutasításához is vezethet. Az élmény alapú tanulás szervezésének legnagyobb veszélye a „megfigyelő” tanuló, mert számára nem tapasztalás ($^{L}2$), hanem ismeretszerzés ($^{L}1M$, $^{L}1P$) történik és utána jellemzően kimarad a saját kipróbálás ($^{L}2$).

Az összes vizsgált taxonómia és a LAU is magában hordozza a tanulás preferált módját: kezdetben befogadó, magasabb szinten alkotó tevékenységet vár el. A kétféle tanulási módhoz

kapcsolódó gondolkodási módszerek többféle megközelítése ismert: a „gyors és lassú gondolkodás” a kreativitás előhívása [26, 27], a természettudományos oktatásban az elfogadó, illetve a felfedező ismeretszerzés [28, 29], a tudáselem megértése, illetve megélése [30, 31]. Mindegyik megközelítésben felfedezhetjük az időbeli tervezhetőség szempontját. A befogadó tevékenység időben jól tervezhető, a LAU^{L}1-5a fázisára jellemző. Az alkotó tevékenység a ^{L}5c fázisban teljesedik ki, de a LAU^{L}4-5 ciklusra szükséges idő egyénenként és helyzetenként változó, kevésbé tervezhető.

A IV/2. mellékletben az egyes gondolkodási és tanulási módokat a LAU-modellben vizsgálom. Az informatikaoktatás szempontjából kiemelt szerepe van a matematikai problémamegoldás [32] a kreatív gondolkodás [33, 35, 36], az informatikai gondolkodás [39] (computational thinking [37, 38]) képességének és az IKT használatának [34]. Ezek „tanulása”, fejlesztése olyan LAU-ok, amelyekben már az első fázisok is alkotó gondolkodást, alkotó tevékenységet várnak el.

A problémamegoldás Pólya-féle [32] és Lénárd-féle [33] leírását, a felfedezést, a kreativitást az időbeliség szempontjából vizsgálva megállapíthatjuk, hogy egy probléma megoldási ideje – legalábbis a gondolkodásfejlesztés szakaszban – nem tervezhető. Ugyanakkor, a hosszú időt igénylő probléma megoldása során bejárt, átélt megoldási (tév)utak is fejlesztik a gondolkodás készségét, teret adnak különböző ismeretek kreatív használatára.

A szakdidaktikai kutatások és eredmények leírhatók LAU-modellben, de hozzá kapcsolódó metodika nélkül nem sokat érnek. A tanítás/tanulás működését metodikai oldalról közelítem meg, tanár–tanuló interakciókat Gordon-módszerrel értelmezve [40]. Tanár a tanítás formájával (frontális, egyéni, projekt…), a személyiségével (magyaráz, demonstrál…), a környezet biztosításával (oktatási eszközök, terem elrendezés…), szabályokkal, fegyelmezéssel, játékosítással motiválja (vagy demotiválja) a tanulót. Ennek a külső motivációs tényezőnek is jelentős a hatása [41]. A tanulás hatékonysága legalább ennyire függ a tanuló belső motivációitól, mentális állapotától. A IV/3. mellékletben az egyéni célok, az erőnlét, az érdeklődés, a remények és félelmek tanulást befolyásoló hatását mutatom be.

Az informatika tanulásának módja kiemelkedő jelentőségűnek tűnik (kifejtve a IV/4. mellékletben), ezért ennek módszertanát [42] a megfelelő oktatási módszerekkel együtt vizsgáltam. Ez az értelmezés teljes mértékben megfelel az oktatásban zajló paradigma váltásnak, amely például az Európai Unió EPSC kiadványában [43] is megjelenik: a tanár, mint a tudás birtokosa és átadója szerep helyett a tanár mentor szerepe válik elsődlegessé.

4 Mi az informatika?

Disszertációm címe: „Az informatika (programozás) oktatásának módszerei”. A téma alapvetően multidiszciplináris, mivel az oktatás (a pedagógia) a humán tudományok körébe tartozik, amelynek eredményeit egy másik tudományterületre alkalmazom. A másik tudományterület – a cím alapján – az Informatika (programozás).

Ezen a ponton a kutatásom során – ahogy a szakmódszertan értelmezésén is – jócskán el kellett időznöm: a kérdésre, hogy mi az informatika, többféle válasz adható, például:

- az informatika a tudományok közül az egyik tudományterület;
- az informatika egy műveltségterület a NAT-ban;
- az informatika egy kompetencia terület;
- az informatika egy szakmacsoport, amit informatikusok gyakorolnak.

Oktatásmódszertani szempontból lenne értelme egy műveltségterület oktatását kutatni, azonban ez a hagyományok nélküli informatika műveltségterület esetén – mint azt a 2020-ban bevezetett NAT-ban szereplő „Digitális kultúra” elnevezés mutatja – nem célszerű. A nagymúltú műveltségterületek jól meghatározott tudományterületekhez köthetők, ezért jelen esetben a változó megnevezésű terület és tantárgy helyett célszerűbb a kapcsolódó tudományterületre fókuszálni.

Az informatika, mint szakmacsoport több oktatási témájú dokumentumban megtalálható. Van ilyen szakképzési ágazat és sokszor láthatjuk tananyagként egyes szakmák, munkafajták, szakképzések felsorolását. Például [7]-ben: programozás, objektum-orientált tervezés… Az informatika ilyen módon történő meghatározása azért nem célszerű, mert a képzések, szakmák, szakterületek listája állandóan változik. Szintén jellemző, hogy az elsajátítandó kompetenciákkal adják meg az oktatás tárgyát. Például „Think like a Programmer…” [44] cikkben az informatikaoktatás (pontosabban Computer Science) részei az absztrakt gondolkodás, a problémamegoldás, a hibakeresés…, de a csoportmunka is szerepel, azaz a szocializációs készségeket is belefoglalja.

Az informatika oktatásáról úgy érdemes beszélni, hogy abban az informatikát tudományterületként értelmezem, de erre sincs egzakt definíció, ezért lehetőség szerint széles forráselemzéssel és szakértőkkel készített interjúk alapján meghatároztam, hogy mit értek informatika tudományon.

Az informatika több neves tudós szerint önálló tudományterület. Az állítást nem az erről szóló tudományos munkák felsorolásával, hanem a napjainkban működő tudományos intézmények létével lehet alátámasztani. A számítástechnikát sokáig a matematikatudomány részének tekintették, amely egyre inkább elkülönül, írta G. E. Fortsythe [45]. Később erre hivatkozva D. Knuth [46] részletesen bemutatja a matematika és informatika tudományok viszonyát. 2003-ban HE Shaoyi Informatics: the brief survey [47] cikkében írta le, hogyan értelmezi az informatika tudományt. Ennek magyar interpretációja Papp Lászlótól „Az informatika fogalma" [48]. A köznapi értelmezést is érdemes figyelembe venni: a Wikipédián szereplő meghatározás [49] forrásai közoktatási és szakképzési tananyagok, illetve szótárak.

Az informatikatudomány a fentiek szerint (továbbá a V. melléklet kiemelései alapján) alapvető, fontos, fejlődő, interdiszciplináris. Shaoyi a tudást, a Wikipédia szócikk az információt nevezi meg az informatika tárgyaként, amelyet kezelni kell. Knuth az algoritmust helyezi középpontba – amely a matematikától megkülönbözteti a számítástechnikát, majd a többi elnevezéssel kapcsolatban vizsgálja, hogy az adatnak és algoritmusnak mi a szerepe.

Az informatikatudomány pontos meghatározása szinte lehetetlen. Annyira interdiszciplináris, hogy nem egyértelmű, magyarázatra, értelmezésre szorul, hogy mi a közös rész. Emellett dinamikusan fejlődik, ezért – azt gondolom – a mibenléte, meghatározása is folyamatosan fog változni, ha a meghatározás a tudományos intézményektől vagy más diszciplínákhoz való viszonyán keresztül történik. Ezért én – némi állandóságra törekedve – azt határoztam meg, hogy mi az, ami szerintem egybefogja az informatika tudományokat, mi a belső közös jellemző és mi az, ami megkülönbözteti a többi tudományterülettől.

4.1 Az informatika tudomány önállósága

Az informatika más tudományokkal való kapcsolata nagyon bőséges kutatási terület, ezért itt csak esetek szintjén foglalkozom a kérdéssel. A kérdés jelen dolgozat szempontjából megkerülhetetlen, mert oktatási szempontból alapvető az informatikát tanító tantárgy(ak) tartalmának meghatározásakor éppúgy, mint a más tudományokat tanító tantárgyakkal való kapcsolat lehetőségeinek vizsgálatakor.

4.1.1 Matematika vs. informatika

Annak ellenére, hogy Knuth 1974-ben részletesen kifejtette, mi a különbség a matematika és az informatika tudomány között [46], a programozást világszerte (így Magyarországon is) tanították a matematika tantárgy keretében. A kérdéskör vizsgálata során oktatókkal, hallgatókkal beszélgetve több esetben hallottam azt az elképzelést, miszerint az informatika a matematika

tudományból ered, annak egyik ága. Egy mérnök hallgató tapasztalata az, hogy alapszinten (kb. 10 éves korig) a két tudománnyal kapcsolatos ismeretek egybeolvadnak, majd ketté válnak, de nagyon magas szinten (egyetemi, PhD) újra egyesülnek. E vélemények relevanciája kérdéses, mivel minden esetben olyan embertől hallottam, aki programok tervezésével, logikai elemzésével foglalkozik, azaz az informatikának a matematikához kapcsolódó ágát látja, éli meg.

Az informatika matematikához való viszonyát – Knuthhoz hasonlóan, de a közoktatás szintjén – megvizsgálhatjuk, ha a két tudományban használt fogalmakat hasonlítjuk össze. Az ebben a témában írt cikkünkben [53] bemutattuk, hogy a matematika és az informatika a két tudomány által egyaránt használt – és a közoktatásban is tanított – fogalmakat másként értelmezi, másként használja. Többek között az „egész számok", mint halmaz számossága és az egész szám informatikai értelmezése más; a „végtelen" matematikában egy absztrakció, míg informatikában nem létezik; a matematika kommutativitás tulajdonságát az informatikában a rövidzár (lusta kiértékelés) felülírja. Programozás szakmódszertani jegyzetben [10 (4.2/J)] az asszociativitásra és disztributivitásra adott példákban láthatjuk, hogy az informatikai véges méretű egész értelmezésnél nem igazak a matematika tételei.

A két tudomány gondolkodásmódjában tapasztalható különbséget egy, mindkét tudományban felmerülő problémán részletesen levezetem a V/1. mellékletben. A vizsgált kérdés a matematika- és informatikaoktatás során sokszor felmerül: természetes szám-e a nulla.

A választól függő további kérdések: Mi a (matematikai) sorozatok értelmezési tartománya? A tömb adattípus első elemének mi az indexe? Hogyan fejezhető ki a páratlan számok összegével az n-edik négyzetszám (a 0^2 is)? Mi a Fibonacci-sorozat első két eleme? Dijkstra – informatikusként – a természetes számok halmazának definíciójához is kapcsolja az érvelését [50], az indexelés 0-ról indítása mellett érvel, de a vita napi szinten zajlik az informatikusok körében. Egy matematika szakdolgozat szerint [51] megállapodás kérdése – nincs vita, ahogy a helyzet adja. –. A „közösségi médiában", a Wikipédián, a hivatkozásokban ellentmondás van (de ez nem zavar senkit).

A probléma matematikai eredetű, de a választ nem a matematika adja. Az adott definíciók (megnevezések) értelmezése alternatív, amelynek következtében módosulnak a tételek. Az értelmezésből adódó modellezés, azaz a specifikáció a kulcs, ezért – azt gondolom – az informatika tudományhoz tartozó problémáról van szó. Az adathoz más értelmezést rendelve módosul az információ, másik modellt kapunk. Felmerül a kérdés, hogy ha a döntés az informatikatudományához tartozik, akkor miért vitatkoznak rajta évtizedeken át az informatikusok? El fog jönni

az az idő, amikor egyértelműen 0-tól indexelnek minden sorozatot? Esetleg lesz egy forradalom, amikor eldöntik, hogy 1-től kell indexelni a sorozatokat? Lehet-e a Dijkstra által megfogalmazott „erkölcsi” támogatás eredménye az, hogy 100%-os dominanciája legyen az egyik értelmezésnek? Azt gondolom, hogy az informatikának nem ez a feladata. Az informatikus mindig az adott specifikációhoz fogja igazítani az értelmezést, ha hiányzik a pontos értelmezés, akkor minden lehetséges értelmezést figyelembe kell vennie. Ha a matematikában véglegesen úgy építik fel a tételeket, hogy a 0 természetes szám és a sorozatok értelmezési tartománya a természetes számok halmaza lesz… az informatika akkor is az aktuális feladat és a programozó számára legkifejezőbb (humán kategória) módon fogja indexelni a sorozatokat, listákat. A matematikai definíciótól, axiómától függetlenül értelmezik az informatikai modellek a 0 szerepét, mindkét értelmezést gyakorlati helyzetekben alkalmazva.[2]

Más tudományok esetén is megfigyelhető, hogy az informatika akkor kap szerepet, amikor egy jelenséghez egy modellt specifikálunk. A jelenségtől, a konkrét esettől függ a megalkotott modell, így a specifikáció is és a modellhez rendelt adatok és ezek feldolgozása is. Ahogy fizikatanári pályám elején tanultam és tanítottam az „Olvasmány egy kíváncsi marslakóról” kitekintést: modellezhetünk egy embert bábuként vagy egérként, attól függően, hogy ruhát vagy egy gyógyszer hatását akarjuk vizsgálni a modell segítségével [52].

Valószínűleg kijelenthető, hogy az informatikának minden tudományban van (vagy lesz) szerepe. A külön tudományként való megjelenését – a számítógép megjelenésével – a tárolt adatok mennyiségének exponenciális növekedése és ezek elemzésére szolgáló eszközök és modellek fejlődése indokolja. Szerepe az egyes tudományokban hasonlít a matematikatudomány szerepéhez, ezért az informatika tudomány önállóságának vizsgálata elsősorban a matematikatudománnyal szemben kérdéses.

Attól, hogy egyes szavak két tudományban szakkifejezésként megjelennek, még nem lesz egyik tudomány a másik része. Például a „parabola” szónak a matematikában és az irodalomtudományban is van értelme, asszociatív kapcsolat van a két fogalom között. Ehhez hasonlót tapasztatam a matematikában és informatikában, például az alábbi két fogalom esetén is:

- A „függvény” a matematikában egy – jellemzően számok közötti – összerendelési szabály (egyértelmű reláció). Informatikában a függvény egy programkód, ami a bemenetén kapott argumentumokat is felhasználva – esetleg mellékhatást is okozva – van, hogy létrehoz a kimenetén egy eredményt. Az informatika függvényfogalma a matematika függ-

[2] A magyar közoktatásban 2020-ban a 0 nem természetes szám, az 1970-es években természetes szám volt a 0.

vényfogalmához képest nem általánosabb, hanem teljesen más a szerepe, mások a jellemzői. Ha kezdetben volt is kapcsolat a két fogalom között, a függvényvizsgálat – egy egyed jellemzésének a fogalmai – informatikai (például komplexitás) és matematikai (például konvexitás) megközelítésben teljesen mások.

- Az objektum orientált paradigma melléktermeke, hogy az osztályok specializációja nem felel meg a matematikai halmazokba soroló osztályozás specializációjának. A legismertebb típus feladat a geometriai objektumok megvalósítása: pont, háromszög, négyszög, sokszög, kör… Matematikában a négyzet speciális téglalap, a kör speciális ellipszis… Informatikában a specializáció során egy objektum leírásához egyedi jellemzőket adunk hozzá. Két adat egyenlősége nem hozzáadható tulajdonság. Fordítva sem igaz: A téglalap nem speciális négyzet, aminek van még egy adata, mert a négyzetben is létezik az adott adat.

A fenti példák a matematika és az informatika alapfogalmait érintik, az ellentmondások azt mutatják, hogy két tudomány fogalmai csak nyelvi szinten közösek. Amikor közös gyökere van a két tudományban egyaránt használt kifejezés fogalmainak, akkor is a fogalom használata, a vizsgálati módszer eltérései miatt nem beszélhetünk arról, hogy az egyik a másik ága, része, beágyazódása.

4.2 Informatika – az egység kétsége

Az informatika tudomány – Couputer Science – az ELTE IK-n a matematika viszonylatában értelmezhető, ahogy azt Knuth és Forsyte is tette. A BME VIK-n viszont a mérnöki tudományokhoz viszonyítva határozzák meg, hogy mit jelent az informatika.

Erre láthattunk példát a *Programozási alapismeretek* (ELTE IK) és a *Programozás alapjai* (BME VIK) tárgyak összehasonlításakor, melynek eredményeit az InfoDidact konferencián cikkben publikáltuk [54]. „Apró” fogalmazásbéli különbségnek látszik a tárgyleírásokban, hogy a programozás folyamatát, illetve gyakorlatát tanítja-e a tárgy. Valójában a különbség a gondolkodásmódban, a „programozás” tevékenység végzésében, a tanított szabályokban is megjelenik. A különbség nem intézményspecifikus, konferenciákon is tapasztalható az eltérés a matematikai, illetve a műszaki szemléletmód szerinti megközelítés között.

- A programozás a matematikából származtatott informatika számára egy fejben elvégzett absztrakció. A programozási feladat megoldása a feladat matematikai absztrakciójával kezdődik, ennek során algoritmikus részekre bontás történik, az egyes részek algoritmu-

sai korábban igazolt megoldásai a részfeladatnak (programozási tételek), ezt zárja az algoritmusnak egy alkalmas programozási nyelven történő kódolása. A programtervező informatikus (Computer Science) ezt a gondolkodásmódot tanulja.

- A programozás a műszaki tudományokból származtatott mérnökinformatika (Computer Engineering) számára egy eszközre írt implementáció. A feladat megoldása az adatszerkezet (objektumok) definiálásával kezdődik, az adatkezelő műveletek implementálása a következő lépés. A feladat megoldása az objektum egyik művelete, függvénye.

A gondolkodásmódbeli különbség nem csak a tanítás során érvényesül. Az oktatókkal beszélgetve kiderül, hogy van, aki szakmája sajátosságának nevezi, mások arra is utalnak, hogy az egyik karon több matematikus végzettségű, míg a másikon több villamosmérnök végzettségű oktató tanít programozást, informatikát. Az ELTE-n szerzett számítástechnikatanári végzettségem nem volt meggyőző a BME-n arra, hogy taníthassak programozást, tapasztalatom alapján okkal. Óralátogatások során megoldottam a feladatokat is és rendszeresen követtem el számomra lényegtelennek tűnő, de az adott helyzetben komolynak számító hibákat, miközben az adott környezetben elfogadottnak tekintett elvi hibákat érzékeltem.

A két szemléletmód közötti különbség akkora, hogy felmerül a kérdés: egyazon tudomány két nézetéről van-e szó. Ebből következően vita tárgya az is, hogy a közoktatásban mit jelentsen az informatikaoktatás: programtervezést (szélsőségként a matematika tantárgyon belül), mérnökinformatikát (csak robotikát) vagy mindkettőt, esetleg még bővebb ismerettartományt.

A gondolkodásmódok különbözőségét a VI. mellékletben példákon keresztül mutatom be.

- A jegyzetekből kiemelt részletek dialektikus elemzése, a pro és kontra érvek nem pártosak, de minden szakmabeli olvasó megjegyezte, hogy az ő gondolkodásmódját jobban előtérbe kellene helyeznem.
- Egy adott feladat – benne eldöntés és keresés részekkel – önkéntes alapú megoldásaiban elemeztük a kódot, hogy hogyan gondolkodnak a programozást oktatók [59].
- Bőséges szakirodalma van a strukturált programozás szabályai alkalmazásának és negligálásának. Az elmélet oldaláról Dijkstra 1968-ban támadja a vezérlés szabálytalan átadását [55], Dijkstra levelének megjelenése után 40 évvel Robert C. Martin a tiszta (jól tervezett) kódról ír a Clean Code – A Handbook of Agile Software Craftsmanship [56] könyvében. A tiszta, szép kódról szóló könyv a releváns kódrészletek 58%-ában szabálytalanul adja át a vezérlést.

Mi az ellentétes szemlélet és gondolkodásmód oka? A kód olvashatósága, értelmezhetősége:

– A strukturált algoritmusban a kód legjellemzőbb részének – a feltételnek – a tagadása szerepel egy logikai AND kifejezés második operandusaként.
– A `break;` vagy többszörös `return` használata mellett a tagadásos összetétel helyett a kívánt feltétel fogalmazható meg, amitől érthetőbb a szándék. Ettől lesz „tiszta”, áttekinthető a kód

Az elméleti megfontolások és a kivitelezés gyakorlata eltérő megoldási módot preferálnak:

– A keresés algoritmusa és implementációja között jellemző eltérés, hogy a programtervező határozottan megkülönbözteti az eldöntést a kereséstől, az eredményt kiértékelve adja meg a választ, míg a mérnökinformatikus a kiértékelést a felhasználóra bízza. Mindkét esetben módszertanilag (szakmailag) alátámasztott a megoldás.
– A programtervező megkülönbözteti az „elemeken végig megyek” és az „elemeken addig megyek amíg” algoritmust. A mérnöki gyakorlatban ez a két megfontolás egybeolvad: „elemeken addig megyek, amíg vannak” megfogalmazás használatával és az adatstruktúrát úgy implementálja, hogy legyen végjel, strázsa…, valami, amiből kiderül, hogy hol a vég.

4.3 Elmélet és gyakorlat

A két egyetemi képzést és gimnáziumi tanórákat elemezve arra kerestük a választ, hogy hogyan kellene a „lineáris keresés”-t tanítani, illetve hogyan tanítják a tanárok, hogyan tanulják a diákok ezt a típusfeladatot [57, 58]. Az eltérő megoldások nem eltérő oktatásmódszertani megfontolásokból származnak, hanem a tudomány (vagy szakma?) eltérő gondolkodásmódjából. Bár a bemutatott példákat oktatási segédletekből vettem, a gondolkodásmódok a mindennapi gyakorlatra jellemzők.

Alsós tanítóknál jól látható, hogy másként írnak a gyerekeknek, mint a hétköznapokban. A felnőtt írás és a betűk írásának tanítása nagyon eltér. Ez a tapasztalat és a programtervező informatikus és mérnökinformatikus képzésben tapasztalt gondolkodásmódbeli különbségek inspirálták annak megvizsgálását, hogy milyen a programozó szakember „kézírása”. Azaz hogyan ír programot az oktató akkor, amikor nem tanít. A két egyetem oktatói önkéntes alapon írták meg egy emelt szintű informatika érettségi programozás feladatának részletét, azt is kommentelve, hogy közben mire gondoltak, mikor teszteltek… 2016-ban az InfoÉRA konferencián bemutattuk, hogy a kódokból egyértelműen látszik, melyik tanszéken, melyik egyetemen dolgozik a kód írója [59]. A tanításkor tapasztalt eltérések a szakma gyakorlása során éppúgy megjelennek. Az oktatás tükrözi a programozásról, a megoldandó feladatról való gondolkodást; szemléletmódot ad át.

Azt látjuk, hogy programtervező informatikus megfogalmaz eljárásokat, algoritmusokat, amiről bizonyítja annak helyességét. A mérnök informatikus pedig a konkrét megvalósításra optimalizált kódot írja.

A BME-n kutatva gyakran előfordult velem, hogy az ELTE-n megismert informatikai fogalmakat újra kellett értelmeznem. Megtanultam, hogy a „számlálós" for-ciklus nem csak látszatra, a nyelvi megvalósítás miatt, de semmilyen szempontból sem számlálós; az adatstruktúra típus, a struct az adott nyelvi környezetben nem egy adattagokat tartalmazó „élettelen" rekord, hanem osztály és osztályként lehet, hogy nincs is adattagja, mert funktor. A programozás a gondolat kifejezése egy adott nyelven, amelyben a nyelvnek különleges szerepe van. Tervezett szabályok mentén fejlődik, mégis, mintha élő nyelv lenne. Míg a tervezés tudományos stílusú, a megvalósítás tájnyelvet használ. Idővel a tájnyelv visszahat a tudományos kifejezésekre, átértelmezi a fogalmakat, megváltoztatja a definíciókat.

A programtervező és a mérnökinformatika az informatikai problémát különböző nézőpontból vizsgálja, sokszor egymást kizáró szempontokat vesznek figyelembe. Jellemző, hogy nem „A" megoldás, hanem egy optimális megoldás a cél, ami a különböző szempontok, különböző súlyú figyelembevételével alakítható ki. Knuth reménye *„A matematikához hasonlóan a számítástechnika némiképp eltér a többi tudománytól, mivel az olyan ember által megalkotott törvényekkel foglalkozik, amelyek bizonyíthatók, a természetről alkotott tapasztalati törvények helyett. …"* Csak félig teljesül az Informatikára. Az Informatika ember által alkotott törvényekkel foglalkozik, de egyben értelmezi is a természet, a környezet jelenségeit, eközben szabályokat alkot. Majd a szabályokat számítógépen implementálja, azaz a természetbe (fizikai környezetbe) visszahelyezi. Úgy tűnik, az informatika tudomány lényege, hogy mindkét módszerrel, a két módszer viszonyával foglalkozik és tudományos eredménye ezek eredője.

4.4 Adat és algoritmus

A programtervező informatikus „elméleti" tervező szemléletmódja és a mérnökinformatikus megvalósításra fókuszáló programozási „gyakorlata" közötti szakadék nem pusztán a probléma elméleti és gyakorlati megközelítéséből adódik. Jellemző, hogy más paradigmák mentén értelmezik és oldják meg a feladatokat a programtervezők és a mérnökinformatikusok.

A programtervezőkre jellemző, hogy algoritmusokon gondolkodnak. Imperatív nyelven programozásban számukra a vezérlési struktúrákból építkező procedurális megoldások jelentik az alapot és a strukturált programozás paradigmáit követik. A vezérlési struktúrákból egyszerű

algoritmusmintákat, ezek kombinálásával egyre összetettebb programozási tételeket, újabb algoritmusmintákat dolgoznak ki, bizonyítják ennek helyességét és sablonként használják egy-egy probléma megoldásához. Jellemző, hogy a problémában szereplő entitásokat egy ID, egy szám jelképezi, az emberek neve épp úgy lehet „1”, „2”… mint a szemének színe. Deklaratív nyelven jellemző a funkcionális programozás preferálása, a megoldás matematikai jellegű, függvénykompozícióként történő megadása.

A mérnökinformatikus az adatimplementációkban gondolkodik. Imperatív nyelven programozásban az egyszerű adattípusokból építkezik, struktúrákat, osztályokat definiálva, az objektum orientált programozás paradigmáit követve. A vezérlési struktúrák az osztálydefiníciók metódusaiban jelennek meg, de összetettségüket nem az algoritmusminták, hanem a létrehozás, a megszüntetés, a saját állapotuk változásának és a környezetükkel való kapcsolatuknak a leírása jelenti. A szoftverek készítése (programozás) során az architektúra adja a szoftver struktúrájának az alapját és kipróbált, tesztelt tervezési minták (design patterns) segítik az összetett problémák megoldását.

A doktori kutatásomban a mérnökinformatikus képzés vizsgálatát az inspirálta, hogy láttam volt tanítványaim megoldását a Nyolc királynő[3] feladatra. Egy olyan backtrack megoldást adtak, amelyben nem ismertem fel az általam ismert backtrack algoritmust. Az VI. melléklet VI/2. példájában egymás mellett látható egy másik feladat különböző paradigmák mentén írt megoldásai. Mindkettő saját kód, a megoldás logikailag azonos, de kognitív modell eltérő, mintha két különböző feladat lenne.

A kétféle paradigma együtt oktatására tett kísérletről olvashatunk az Egy webes játék készítésének programozásdidaktikai szempontjai című cikkben [60]. Az eredeti cél, hogy a programozási tételeket játékos környezetben vezesse be, csak nagy kerülővel lehetséges. A webes OO környezetben először az entitásokat, az objektum absztrakciókat, sprite-okat, grafikus keretrendszert kell megismerni, ezután jöhetnek az ezekből alkotott sorozatok, amelyek kezeléséről a módszeres programozás szól.

Ugyanannak a feladatnak a megoldására a különböző programozási paradigmák más-más absztrakciók alkalmazásával hatékonyak. Az alacsonyszintű, processzor számára írt programokban a goto használata jellemző, hiszen a processzor az elágazások és ciklusok helyett csak ezt ismeri. A közoktatás programozásoktatásakor is gyakran felmerülő viták, például break; használatáról, a végtelenségig folynak, mert a paradigma és az absztrakció választása van az eltérő nézetek hátterében. A break; esetén az OOP rövid, elemi algoritmusaiban fontos, hogy a

[3] Helyezzünk el a sakktáblán 8 királynőt (vezért) úgy, hogy egyik se tudja ütni a másikat.

vizsgált feltétel legyen olvasható és ne a tagadása; ez indokolja a használatát. A módszeres programozás az elemi algoritmusokból összetett algoritmusokat képez, ahol a vezérlési struktúrák megszakításának következménye nem egyértelmű, ezért tiltja a használatát.

Az Informatika tudományban alapvető, ezért tanítása során hangsúlyos kell legyen, hogy minden esetben a feltételeket, környezetet, szempontokat meg kell vizsgálni, a döntéseket ezek alapján kell meghozni, a megfelelő absztrakciókat és hozzájuk tartozó hatékony eszközöket ennek megfelelően kell alkalmazni. Az elvek, a tételek, a tapasztalat, a minták nem alkotnak statikus rendszert, ezek alternatív eszközök különböző problémák értelmezéséhez, megoldások alkotásához.

4.5 Az Informatika fogalmai

Nemcsak az elvek és a tételek változnak dinamikusan. A problémamegoldás során az új megoldások alkotása a fogalmak értelmezésének állandóságát is kikezdik. Ez a jelenség – úgy tűnik – sokkal gyakoribb, mint más tudományoknál. Az VI/3. mellékletben néhány példa található arra, hogy egy-egy kifejezés csak az adott környezet, az alkalmazott modell pontos ismeretében értelmezhető.

Egy új tudomány – tudományág – esetén természetes, hogy a használt fogalmak alakulnak, kiteljesedik az értelmezésük. Azonban azt gondolom, hogy itt nemcsak a kiteljesedést éljük meg intenzíven. Informatikában a fogalmak a használat során, a problémákhoz, a nyelvekhez, a modellekhez igazodva nyernek új értelmet, a definíciók domain függők.

4.6 Az Informatika esztétikája – szépség és tisztaság

Ami igaz, az természetes, ami természetes, az jó és szerintem szép is. (Petőfi)

Az informatika, mint tudomány a matematika és műszaki tudományokhoz áll közel. Azonban Dijkstra tudományos munkásságának feldolgozására [61] és a Clean Code [56] könyvbéli kód tisztaságának jellemzése is inkább egy humán típusú tudomány kérdésköréhez tartozik. A gépnek vagy a természetnek mindegy, hogy strukturált-e, tiszta-e a kód. A kód olvashatósága, megfelelő változónevek választása, a dokumentálás, a modell-alkotás, az adatábrázolás… sorolhatjuk azokat a témákat, amelyek leginkább az ember szellemi tevékenységével kapcsolatosak, humán vonatkozásaik vannak. Dijkstra levelei nem csupán tudománytörténeti érdekességek, hanem az informatika tudományos eszköztárának meghatározásához járulnak hozzá.

Az informatika, a digitális világ elvileg szorosan kapcsolódik a gépekhez, az elektronikához. Mégis, humán vonatkozásai egyre erőteljesebbek. A kódot nem(csak) a gép számára írjuk. A

2017-es InfoEra konferencián elhangzott „Melyik rendezés? – Kódolvasási készségek fejlesztése” előadásból [62] idézve:

> *when you program, you have to think about how someone will read your code, not just how a computer will interpret it. (Kent Beck)*[4]
>
> *A kódot többször olvassák, mint ahányszor megírják!*
> *A programkód olyan, mint egy DVD: egyszer megírják, nagyon sokszor olvassák*

Mint humán által a humán olvasónak írt szöveg, a kódnak is van stílusa, amely jellemző a gondolkodásmódra, a programozó egyéniségére. A gép végrehajtja a program utasításait, a programot olvasó értelmezi azt és környezetével, megfontolásokkal együtt próbálja meg átvenni a kódolt gondolatot.

Tanárként, az oktatás módszerének és gyakorlatának tervezéséhez és kivitelezéséhez elengedhetetlen a minőségi tudás. Írástanításnál a szép írás, nyelvnél a helyes kiejtés, matematikában a gondolatmenet elegáns leírása, fizikában a precíz mérés és adatgyűjtés. Informatika oktatása során is fontos nevelési-oktatási feladat a „szép” megoldások tanítása, ami a megoldáshoz kapcsolódó megfontolásokat is tartalmazza. Programozás tekintetében nagyon nehéz megmondani, hogy milyen a szép megoldás. Matematikai és műszaki kérdés a hatékonyság, ezen kívül, a humán tudományok tartományába tartozik a „szépség” és az „esztétika”.

Egy-egy feladat megoldásakor felmerül a kérdés, hogy mely nyelvi elemek használata „illik” a feladathoz. Melyik megoldás szebb? Melyik jobb? Mennyire bízzuk a kódunkat a fordítóprogram optimalizálási képességeire? Kutatásom, az oktatás módszerei szempontjából: Melyik megoldási módot tanítsuk? Melyik elveket alkalmazzuk? Melyik nyelvi subset ad szebb megoldást? Ha az egyik megoldási módot példaként bemutatjuk, akkor a másikat hogyan értékeljük? Mely érvek fontosak? Mit kezdjünk az egy adott környezetben helyes, de egymásnak ellentmondó érvekkel?

A kutatás során sokszor külső, majd belső konfliktusaim voltak a „szép”, másként a „jól olvasható” megoldással kapcsolatban. A VI/4. mellékletben kigyűjtöttem néhány tipikus esetet, amelyekben a kód várható olvasóinak függvényében adtam megoldást (ha tudtam). Emellett

[4] Fordítása: Amikor programot írsz, arra is gondolnod kell, hogy valaki olvasni is fogja, és nemcsak arra, hogy egy számítógép hogyan interpretálja.

olvasható egy konkrét eset elemzése: az Ötszáz nevű érettségi feladat[5] fizetendő összeget kiszámító függvénye, amelyet az Igy írtok Ti… – korábban már említett előadásunkon [59] Czirkos Zoltán elemzett mérnöki megvilágításban a „szépség” szempontjából.

A kódolási stílus sok szempontból egyéni, tükrözi a gondolkodásmódot. A kód esztétikai minősítése függ az éppen használt nyelv specialitásaitól, a felhasználás tervezett környezetétől, a programozó korábban megoldott feladataitól (azaz szakmai tapasztalatától) és kódolási szokásaitól. A kóddal szembeni elvárásokat humán szempontok is (lehet, inkább főleg humán szempontok) alakítják. Az informatikai cégeknél a projekteken együtt dolgozók közös stílust kell, hogy kövessenek, mintha egyforma lenne az ízlésük.

Tovább lépve a programozás alapjairól, ugyanilyen szubjektív, humán szempontokkal tűzdelt a programozási nyelvek minősítése, használhatósága is. Melyik könnyű, tiszta vagy átlátható? Mérésekkel, rangsorokkal igyekeznek objektív minősítést adni, de amit mérnek, az szubjektív, jelentősen befolyásolja a divat. A más funkciójú, azonos célú szoftverek, operációs rendszerek minősítésére is jellemző, hogy a nagyon konkrét informatikai érvek kizárják egymást és a megoldás választásánál a megszokás vagy az ízlés dönt.

Mindezek alapján elmondhatjuk, hogy az informatikai megoldások minősítésében a matematikai bizonyításon és műszaki megfelelésen túl, esztétikai – humán – szempontokat is figyelembe kell venni.

4.7 Az Informatika

Az informatika tanításának módszereit vizsgálva, a második alappillér és ezért nélkülözhetetlen **az informatikatudomány** (a tanítás tárgyának a) **meghatározása**. Az informatikusok képzése és az informatika más tudományokkal való kapcsolata alapján olyan meghatározást szeretnék adni, amely legalább néhány évig releváns, benne van az, amivel az informatikusok – a tudomány művelői – foglalkoznak és útmutatást ad identitásra, a többi tudománytól való megkülönböztetésre éppúgy, mint az azokkal való kapcsolatra. A fejezetben fentebb leírt, informatika tudomány jellemzésére vonatkozó megállapítások alapján született az alábbi specifikáció[6]:

- Az informatika alapfogalma az **adat**.

[5] https://www.oktatas.hu/bin/content/dload/erettsegi/feladatok_2016tavasz_emelt/e_infmeg_16maj_ut.zip [2021.10.30]

[6] T1 tézis első felével kapcsolatos elemzések összefoglalása

- Az **információ** az **értelmezett** adat. Az információ **tudás**, az értelmezés **gondolkodási folyamat**. Az információ újabb adatok létrehozásának forrása (adat-termelő), ez adja dinamikáját.
- Az informatika az adat/információ létrehozásának, tárolásának, továbbításának, módosításának, rendszerbe szervezésének a tudománya.
- Az informatika legfontosabb fogalmai: **algoritmus (szekvencia, alternáció, ismétlés), objektum, komponens, rendszer, modell**. Ezek a fogalmak **absztrakció**val adatként megjeleníthetők és értelmezhetők, a köztük levő kapcsolatok is adatok.
- Az értelmezés algoritmikus, azaz szekvenciák, alternatívák, ismétlések folyamata. Az értelmezés ezért többféle eredményre, **alternatív megoldásokra**, tudásokra vezethet.
- Az informatika művelését, a megoldásokat **hatékonyság**gal, **pontosság**gal **relevancia**val minősítjük.
- Az értelmezés – mint gondolkodási folyamat – megvalósítása az **informatikai gondolkodás** (computational thinking: J. M Wing [37, 38], magyar kifejezés Pluhár [39]).
- Az informatika gyakorlati tevékenysége a **problémamegoldás**, produktumelőállítás, reprodukció.
- Az informatika **véges méretű** adatokkal, struktúrákkal, modellekkel, rendszerekkel foglalkozik.
- Az informatikában a gyakorlat nem feltétlenül igazolja az elméleteket, hanem **átértelmezi az aktuális problémára**, vitatkozva, **ellenőriz**ve az alkalmazhatóságot.
- **Az informatikatudomány a rá jellemző gondolkodásmódjával, problémamegoldási stratégiájával, megoldási módszereivel** járul hozzá a világ megismeréséhez, ezért **önálló tudomány**, de hatását nagymértékben fejti ki interdiszciplináris területeken. Ezért **„alaptudomány”**.
- **Az informatikatudomány** mindig születőben, újjászületőben lesz, **megújulását az adat és az információ dinamikus kezelése (az értelmezés) belső forrásként tartja fent.** A tudomány fogalmai új értelmezéseket kapnak, részterületei folyamatosan átalakulnak.

Az Informatika besorolása a tudományok közé rendkívül nehéz, mert alapjaiban érinti a matematikát is, reáltudományokat is és egyre inkább humán vonatkozásai is előtérbe kerülnek. **Az Informatikatudományon belül az elméleti megfontolások és a gyakorlati tapasztalatok, a különböző paradigmák**, mint Yin és Yang, **egymással sokszor szemben állóan, egyenrangúan, de mégis egymást kiegészítően kapcsolódnak egymáshoz. A különböző megoldási stílusok az emberi gondolkodás, az értelmezés, az absztrahálás sokszínűségét mutatja.**

4.8 A Programozás

Ahogy az informatikatudomány definiálása sem egyszerű, programozáson is sokan, sokféle dolgot értenek, melyek némelyike ellentmond más értelmezéseknek. **A programozás** fogalmának leírása:

- A programozás fogalmához mindenképpen kapcsolódik a **kódolás**, azaz a gondolat megjelenítése, amihez valamilyen nyelv, közvetítő médium szükséges.
- A programozás **kommunikáció** a géppel és a program többi olvasójával, alkotójával.
- A programozás tevékenység, amelynek során **adatokat, algoritmusokat, modelleket** úgy **írunk le**, hogy az **a gép számára** közvetve (vagy közvetlenül) **értelmezhető legyen és meghatározza annak működését**.
- A programozás az informatikai gondolkodás **implementáció**ja egy gépre.
- A programozás a nyelv és a **médium használatának, a kommunikációnak a művészete**. A programozási nyelv mesterséges, nyelvtana **matematikai gyökerű**.
- A programozás gépi implementáció, ezért **műszaki tudomány**.

A programozás elválaszthatatlan része az Informatikatudománynak, de az Informatikatudomány több, mivel nemcsak a géppel megoldható problémákkal foglalkozik. Ugyanakkor jellemző, hogy az informatikai problémák programozással modellezhetők számítógépen.

Napjainkra jellemző, hogy az adatokat digitálisan tároljuk, a programok fizikai megjelenése digitálisan tárolt adat, így **a programozás a digitális világ megismerésének és alakításának eszköze**. Mivel történtek kísérletek analóg számítógép készítésére és programozására, **sem a programozás, sem az Informatikatudomány nem korlátozható le a digitális világ tudományára**.

5 Az informatika tanításának tantárgya

Kutatásom harmadik alappillére az informatikaoktatás szervezeti kereteinek vizsgálata, az a keretrendszer, amiben az megvalósul.

A számítástechnika tudománnyá válásának első pillanatától kezdve igény volt a téma közoktatás keretében történő tanítására. Országonként, korszakonként, tantervenként más-más formában jelent meg a közoktatásban a téma.

Egyetemek szervezeti változásain látszik, ahogy tudományáganként kiválik az informatika. Napjainkban Magyarországon a két fő egyetemi képzési terület a programtervező és a mérnökinformatikus, de gazdasági informatikus, médiainformatikus, könyvtárinformatikus szakok is indulnak. Vizsgálat tárgya lehet, hogy ezeket az egyetemi szakokat meg kell-e jeleníteni a közoktatásban, ha igen, akkor integrálva a megfelelő tantárgyakba vagy külön informatika tantárgyként. A kérdéskört realizálva megvizsgáltam, hogy a magyar tantervekben hogyan jelenik meg az informatika oktatása, illetve optimális esetben hogyan kellene megjelennie. [7]

- Mi az, amit informatikából nem az erre fókuszáló tantárgyban lehet tanítani?
- Mit kell informatikából egy erre specializált tantárgy keretei között tanítani.
- Melyek azok a témák, amik – bár nem tartoznak az informatikatudás körébe –, az informatikát tanító tantárgyban lehet (vagy kell) tanítani?

A válaszok jellemzően alternatív lehetőségeket kínálnak, de a konkrét válaszokból következik, hogy szükség van-e informatika tantárgyra. Ha igen, akkor meghatározható, hogy mikor, mennyi időben kell tanítani.

Az informatika oktatásához tanár is kell. Ezért az eredményből következik, hogy az informatikatanárnak – aki az adott tantárgyat tanítja, így a tanított területek minden részéről legalább alapszintű ismeretekkel kell rendelkeznie –, mit kell tudnia; valamint mely témákban kell tájékozottnak lennie arról, hogy a tantárgyon kívül hogyan egészítik ki és használják az informatikai ismereteket. Egyúttal az is következik, hogy az informatikát más tantárgy kereteiben tanító szaktanárnak milyen informatikai (szak)ismerettel és informatikaoktatási módszertani ismerettel kell rendelkeznie.

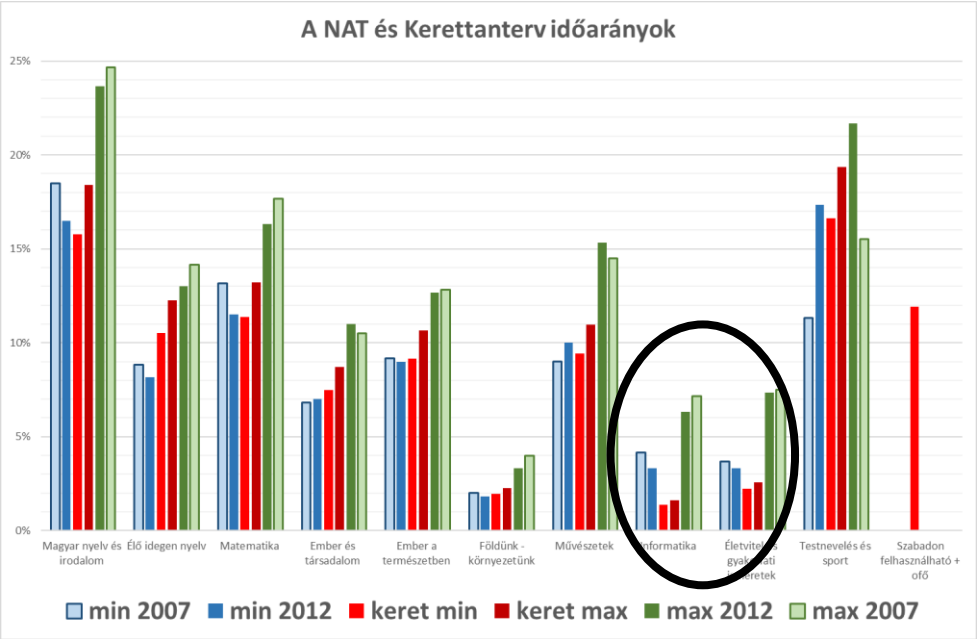
A VII/1. mellékletben összefoglaltam a tanterveknek és ezek oktatásához készített tankönyveknek az informatika tartalmak megjelenése szempontjából végzett elemzéseket. A magyar informatikaoktatás kezdeteitől visszaemlékezés [63] formájában majd az 1995-ös [64], a 2003-

[7] Ebben a fejezetben és a kapcsolódó mellékletekben és cikkekben az elemzéshez a T3 tézisben hivatkozott LAU-modellt használom.

as [65], a 2007-es [66], a 2012-es [67] és 2020-as [68] Nemzeti alaptantervekben, valamint a 2013-as [69] és 2020-as [70] kerettantervekben nyomon követhető, hogy az elmúlt körülbelül 40 évben mit tekintettek számítástechnika majd informatika, legújabban digitális kultúra ismeretnek és ez az ismeretkör hogyan jelent meg az egyes oktatásszervezési dokumentumokban. A követelményekhez mindig készültek tananyagok, tankönyvek, amelyekből láthatók a kivitelezésre vonatkozó elképzelések. A tucatnyi tankönyvsorozat és feladatgyűjtemény közül néhány a tantárgyi megközelítés szempontjából is releváns. Az elemzés alapján elmondható, hogy

- a tantervekben mindenhol az informatika általam (4.7 fejezet) adott meghatározásának egyik-másik (de nem minden) pontját emelik ki az informatikaoktatás feladatának;
- a technika tantárgyon belülre szervezett informatikaoktatás [71] esetleges volt;
- a matematikatantárgyon belülre szervezett informatikaoktatás [72] elmaradt;
- az újabb NAT-ok egyre növekvő mennyiségű tananyaga teljesíthetetlen, helyette a kompetenciákat kellene előírni.

A 2012-es NAT [67] és 2013-as Kerettanterv [69] a korábbihoz képest minden műveltségterülethez több tananyagot rendelt. A túlméretezés kivédésére két műveltségterülethez rendelt idő egy részét más tantárgyakba integrálták. Így az Informatika és az Életvitel és gyakorlati ismeretek műveltségterület a NAT-beli arányoknak megfelelő óraszámnál kevesebb órát kaptak a Kerettantervben (a 3. ábra bekarikázott részén a piros oszlopok a szomszédjaiknál alacsonyabbak). A 2013-as kerettantervben – 1–12 évfolyamokra összesítve a heti óraszámokat – 5 informatikaóra (~1,5%) jutott, a 2007-es és 2012-es NAT százalékos arányai heti 16 órának (~5%) felelnek meg. Zsakó László 2015-ben a „Miért elavult informatikából a NAT, a kerettanterv, az érettségi” [73] prezentációban foglalta össze a problémákat, amelyek alapján a 2020-as NAT-ra [68] vonatkozóan a tartalom struktúrájára és óraszámokra ajánlásokat fogalmazott meg [74]. A 2020-as NAT-ban a struktúra módosítás megvalósult, de javasolt, és a korábbi NAT-ok százalékos arányainak is megfelelő 16 óra helyett csak 11 óra (~3,5%) lett.



3. ábra: A 2007-es és 2012-es NAT, valamint a 2012-es Kerettantervben a műveltségterületekre meghatározott időarányok

A számok közötti eltérés feloldására papíron megoldási lehetőség a tananyagsűrítés, illetve a tantárgyon belüli és tantárgyközi integráció. A tananyagsűrítés a tankönyvben jelentős tananyagcsökkentés és felszínes ismeretekátadás formájában realizálódott [75], egyes évfolyamokra egyáltalán nem készült tankönyv.

A VII/2. mellékletben bemutatom, az ISzE 2015-ben végzett kutatásának eredményét [76], amely a 2013-as kerettanterv megvalósulását vizsgálta, illetve ennek LAU-modellel történő értelmezését. A felmérés és a mindennapi tapasztalat is azt mutatja, hogy ha a tantárgyi integrációban egy LAU └1–5a fázisai szerepelnek, akkor a hozzátartozó időre is szükség van. A └5c, esetleg └5b fázis az, ahol komplex – más tantárgyon belül történő – felhasználással lehet időt nyerni. Az 1/3-ra tömörített informatikaórán – ahogy az a „Why Can't I Learn Programming?" The Learning and Teaching Environment of Programming cikkünkben [77] olvasható – az egyes témák és kompetenciák └1 fázisra sincs idő, jellemző, hogy a └2 fázis a házi feladat. A következtetéseket nemzetközi kitekintéssel is alátámasztja Torma Hajnalka írása [78].

A magyar szabályzók mellett igyekeztem más országok tanterveit, informatika óraszámokat és tananyagmodulokat megismerni, ezeket az elemzések, jelenségek értelmezése során felhasználni. A tantárgyként, a más tantárgyba integrált vagy választható tantárgyként történő oktatási formák alkalmazása országonként változó. Vannak országok, például Finnország [79], Észtország, ahol a tantárgyi integráció nagyon erős, ahol nincs, vagy alig van nevesített informatikaóra. Máshol, például Nagy-Britaniában [80] a 2014-től érvényes tanterv szerint minden évben

van informatikaóra. Közvetett tapasztalatokat jelentett számomra az ISSEP konferenciák többek között francia [81], holland [82, 83], lengyel [84] és német [85, 86] informatikaoktatásról szóló elemzések. Emellett testvériskolai, diákcsere és projekt kapcsolatok révén iskolalátogatással személyes élményeket is gyűjtöttem a szlovák [87], az erdélyi [88], az indiai [89, 90] és a maláj [91] oktatási rendszerről, informatikaoktatásról.

5.1 Tantárgyi integrációs tapasztalatok Informatikából

Elvileg elképzelhető, hogy az informatika más tantárgyakba integrálásával időhatékonyabb az oktatás, hiszen hasonló elv alapján érdemes két-tannyelvű képzésen részt venni.

Az, hogy a magyar közoktatásban az informatika tartalmakat nem sikerült más szaktárgyakba integrálni, nem jelenti azt, hogy nem is lehet. Az integrálás sikerességének jól látható szempontjai:

1. Milyen az eszközellátottság. Jut-e mindenkinek, minden órán gép, hálózati kapcsolat.
2. A különböző szakos tanárok mennyire felkészültek informatikából.
3. Milyen oktatási formák valósíthatók meg. Például projektoktatás, flipped classroom.
4. Melyek a konkrét integrálandó témakörök.

Az első finanszírozástól függ, a második a tantervhez igazodó tanárképzéssel és továbbképzéssel valósítható meg. Az oktatási formák és az oktatásmódszertan jelentősen módosítja az integráció értelmezését is, ezért ezt külön fejezetben (7. fejezet) vizsgálom. A szaktárgyi oktatás keretében megvalósítható integrációs lehetőségek részletes elemzése a VII/3. mellékletben olvasható.

A mellékletben áttekintem és az integrációs lehetőségek szempontjából elemezek szakmai ajánlásokat [92, 93, 94], szakpolitikai stratégiákat, jelentéseket [95, 96, 97]. Az informatika műveltségterület[8] tartalma az idők folyamán csak az eszközök, illetve szoftverek fejlődése miatt változott, bővült. Jellemzőbb a hangsúlyok eltolódása, a minimum elvárások szintjének emelkedése. Ezzel együtt az integrálásra ajánlott témák is viszonylag kis mértékben változtak. Ezért a megállapításokat a legutóbbi, 2020-as NAT [68] témakörei alapján foglalom össze. Az informatika oktatására szánt tantárgy neve itt Digitális kultúra, az informatikai tartalmak négy fő területbe lettek sorolva:

1. Az informatikai eszközök használata
2. Digitális írástudás (szöveges, rajzos, táblázatos dokumentumok, internetes kommunikáció)

[8] Itt az informatika műveltségterületről nem feltételezem, hogy az általam specifikált informatikának megfelel, a két fogalom összehasonlítása külön lesz.

3. Problémamegoldás informatikai eszközökkel, módszerekkel (összetett problémák megoldása, algoritmusok, adatmodellek, adatbázisok, táblázatkezelés, robotika, programozás)
4. Információs technológiák (webes- és mobiltechnológiák)

5.1.1 Az informatikai eszközök használata (1.)

A tanítási gyakorlatot is figyelembe véve, az eszközök használatát a megfelelő LAU-okba beágyazottan kell (lehetne) tanítani. A beágyazott LAU azonban nem lehet nagy, praktikusan 5 percnél több időt nem vehet el a tanóra fő tananyagából. Így integrálható egy jól használható tantárgyi oktatóprogram használatának megtanítása, joystick vagy mérőeszköz használata. Nem integrálható a billentyűzetkezelés. A gépírás modulként és 3-5 tantárgyba beépítve is tanítható, de mindenhol alapvető, hogy rendszeresen időt kell rá szánni. A beépülés valójában közbe-ékelés, emellett a helyes rutinok elsajátítása érdekében a szaktanár képzettsége is szükséges. Ugyanez a helyzet a robotok, automaták, digitális laborok esetén is.

5.1.2 Információs technológiák (4.) integrált oktatása

Bár ez sorrendben a negyedik téma, a más tantárgyakba integrálhatóság szempontjából az eszközök használatánál leírtakhoz nagyon hasonló. Tartalmát tekintve ebbe a témakörbe tartoznak a tanórákon használható kommunikációs, tanulásmanagement és oktatószoftverek, mobil és webes applikációk (pl. Geogebra [98]), a számítógépes mérés-kiértékelés, internetes keresés. Az egyes technológiák (illetve a szoftver új funkcióinak) szaktantárgyba integrált oktatás LAU-jára legfeljebb 5 perc/tanóra fordítható. Jellemző megvalósíthatósági feltétel, hogy a tanárnak sokkal jobban kell ismernie a technológiát, mint a diákoknak, hogy a használat közben felmerülő technológiával kapcsolatos problémák megoldása is beleférjen az 5 percbe.

5.1.3 Digitális írástudás – informatika alkalmazói ismeretek – (2.) integrált oktatása

A közoktatásban azokat az általános alkalmazás-típusokat érdemes informatikaóra keretében oktatni, amelyek használata a tipikus élethelyzetekben jellemző, nem tantárgy (vagy szakma) specifikus. Az informatika tantárgyon belüli megjelenésük oka az egyes alkalmazások komplex felhasználási lehetőségei. A szoftver alapfunkcióinak használatán túl az alkalmazástípus jellemzőit (jellemző objektumait, struktúráit, eljárásait és tulajdonságait) is ismerni kell a hatékony használathoz. Itt a tantárgyi integrációnak az egyes alkalmazás típusok ᴸ5b–5c fázisában van haszna, az integrációt megelőzően(!) kell – informatikaórán – teljesíteni a ᴸ1–5b fázisokat.

5.1.4 Problémamegoldás informatikai eszközökkel és módszerekkel (3.) integrációs lehetőségei

A témakör egyes témáit sokáig szakmai tudásként tartották számon, amelynek ismeretére az átlagos felhasználónak nincs szüksége. Integrált oktatására nem találtam hatékony megoldást, ezért ez tekinthető az informatikaoktatás központi témájának. A megvalósuló integrációk tantárgyközi projektként vagy modulként valósulnak meg, ami a magyar közoktatásra nem jellemző.

5.1.5 Tantárgyi tartalmak beépítése az informatika tantárgyba

Az informatika tantárgy léte azért állandó kérdés még napjainkban is, mert a tanított tartalmak jellemzően más szakterületeket is érintenek. A tantárgy léte napjainkban azon alapul, hogy mely tartalmak nem taníthatók más tantárgyon belül, melyek azok a „21. századi", „digitális", „számítógépes" ismeretek, amelyeket a többi szaktantárgy készen szeretne kapni. Ez a szemlélet vezetett az egérkezelés, a menü tanításához, a más tanórákhoz „házi feladat" elkészítéséhez és tette sikertelenné az informatikaoktatást. Az informatika tantárgyba (akkor is, ha épp digitális kultúrának hívják) integrálható tartalmakra ugyanaz a szabály érvényes, mint a fordított irányra: Időkeret biztosításával modul-szerűen lehet integrálni olyan tartalmakat, amelyekre az informatika tartalmak épülnek (ide tartozik például a gépírás, esetleg robotok építése) vagy tanóránként 5 percben tárgyalható. Az integrált tananyag ebben az irányban is elsősorban a másik tárgyban tanultak ᴸ5b–5c fázisú kiegészítése [99]. Ez a kritérium például a robotika esetén az elektromos áramkörök, matematikából a megoldóképletek ismeretét feltételezheti. Különösen fontos azoknak az ismereteknek a megfelelő beépítése, amelyek tantárgyként nem jelennek meg, például: etika, ergonómia.

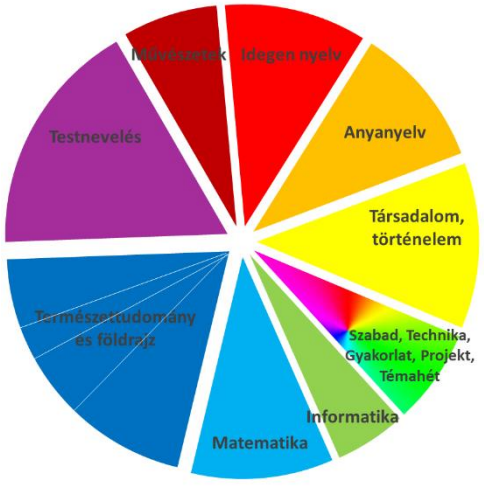
5.2 Tantárgy és a projekt

A szaktantárgyi kereteket megbontó projekt alapú oktatás a tantárgyi integráció teljesen más formáját teszi lehetővé [100]. Magyarországon ilyen jellegű oktatás jellemzően alapítványi vagy magán oktatásban létezik[9]. Emellett vannak törekvések arra, hogy a szaktantárgyi keretekbe is beillesszék a projekteket, általában a tanév egy hetében. A projektoktatás és ezen belül az informatikaoktatás lehetőségeiről a VII/4. mellékletben olvasható elemzés. A szaktantárgyi, tanterv alapú oktatás tervében a LAU-ok első fázisain és egymásra épülésükön van a hangsúly.

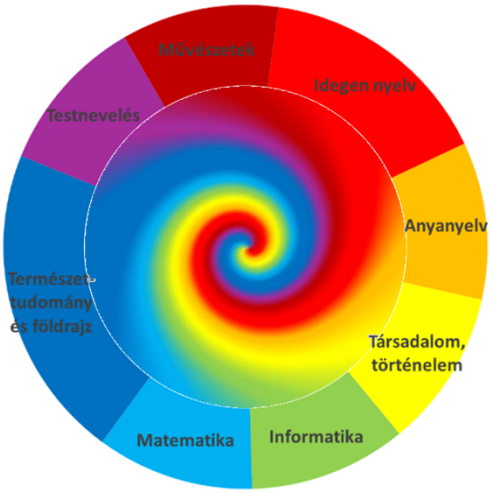
[9] Például Budapest School (https://www.budapestschool.org/hu/) [2021.10.30]

A tudás minőségi fejlesztésére sokszor nincs idő tervezve, illetve gyakorlásként, fiktív idő szükséglettel, házi feladatban jelenik meg. [bővebben 78] A projekt módszer szakirodalma [101, 102, 103, 104, 105, 106, 107] alapján, a jó projekt tervezése az elérendő célból indul, LAU^{L}3-5c fejlesztést tervezi, az első fázisok önálló tanulással teljesítendők. Ebben hasonlít a flipped classroom módszerhez, amelyben a LAU^{L}1-3 otthoni, önálló munka, míg a LAU^{L}3-5c tanórai, tantárgyi munka, de a projekt esetén a tanulás helyszíne nem ennyire meghatározó.

Amennyiben a projekt tantárgyhoz kötődik, akkor a LAU-ok tervezésénél az ^{L}4 fázis (idő) két szempontból is fontos. Egyrészt, az ismereteknek idő kell az elraktározódáshoz; nem hatékony a túl gyors haladás, mert közben nem várható áttérés az ^{L}5c szintre. Másrészt, ha „csomagban" jön az informatika, akkor utána a más témák csomagjai miatt hosszú kimaradás lehetséges; dominánssá válhat a felejtés. Ezért az igazi projekt alapú oktatásban nem a „formabontó napok", hanem a tantárgyköziség a fő jellemző. A valódi projektoktatás a teljes tananyag szaktárgyakra bontása (4. ábra) helyett a műveltségterületek kereteivel támogatott komplex képzést ad (5. ábra), amiben eltűnnek a tantárgyi órák határai, ahogy azt a finn tanterven látjuk [79].



4. ábra:
Tantárgyak időszeletei a 2020-as NAT tervezetben[10]



5. ábra:
Tantárgyak időszeletei projektoktatáshoz[11]

Így valósíthatók meg a STEM vagy STEAM[12] projektek is. A projektet vezető/felügyelő tanárnak nem kell rendelkeznie az egyes tantárgyak szakismereteivel, de a háttérben biztosítani kell a szakmai vagy szaktanári mentorálást, a „távoktatást" vagy konzultációs lehetőséget. Az eredmények minősítése, az értékelés, az egyes tantárgyak (tudományok) keretén belül történik.

[10] Saját készítésű ábra. A 2020-as NAT óraszámai alapján készült.
[11] Saját készítésű ábra. A tantárgyi órák és a projektoktatás integrálása.
[12] STEAM: Science, Technology, Engineering, Arts, Mathematics (STEM: u.ez, „arts" nélkül)

A tanév rendjében meghirdetett „projekthét” vagy „projektnapok” komplex tanulási terek, de időtartama és akció jellege miatt az éppen aktuális LAU-ok nem kapcsolódnak szervesen a tanulmányokhoz. A projekteknek az „egyéb” pedagógiai hasznát szokták kiemelni, a projektmódszer tanulása adja a pedagógiai értékét, amiben szórványosan az informatikai ismeretek L5c szintű használata is kimutatható, de jellemzőbb cél az érdeklődés felkeltése, egy kutatási vagy együttműködési módszer kipróbálása.

A digitalizáció és az információs robbanás szükségessé tették az oktatási módszerek és tartalmak teljes átalakítását, a kompetenciákra, kreativitásra, együttműködésre helyezve a hangsúlyt. A projektoktatást nagyban segíti, támogatja a digitalizáció és mindkettőnek szükségszerűen be kell épülnie a közoktatásba. De a projektmódszer alkalmazásához nem szükséges digitális írástudás és a digitális írástudás fejlesztése önmagában nem követeli meg a projektmódszer alkalmazását.

Az informatika oktatása ennek ellenére – tantárgyi keretek között is – hatékonyabb, ha projektekre épül. Ennek oka az, hogy központi szerepe van a tananyagban a problémamegoldásnak, ami az egyes helyzetekben célhoz – a probléma megoldásához – rendeli a szükséges ismereteket. A projektoktatást motiválja, hogy a 2020-as kerettantervben néhány témakör óraszáma értelmezhetetlenül kevés, ha egyedi tanóraként tekintünk rá, csak komplex – projekt jellegű – feladatok részeként értelmezhető. Ebből következik, hogy – bár a NAT, ebből a kerettanterv és a helyi tanterv a szövegezésében következetes – a jó tanmenetekben és a tanórák címében valamilyen (mini) projekt van megnevezve, nem a tananyag. (Például: „Meghívó készítése” a „Betű és bekezdés tulajdonságai” helyett.)

5.3 Az informatika szerepe a közoktatásfejlesztésben

Az informatika nem csak tananyagként jelenik meg a közoktatásban, hanem az oktatás módszerei, tanári kompetenciák révén is. A projektoktatás, a távoktatás, a fordított osztályterem esetén kihagyhatatlan az infokommunikációs technológiák, digitális eszközök alkalmazása. A 21. század informatikai stratégiái mind kiemelten foglalkoznak az oktatás szerepével [108, 109, 110, 111] 2016-ban az oktatás átalakításához a Digitális Jólét Program részeként külön stratégia is készült [112] ami kapcsolódik az európai stratégiához [114] és a kivitelezését alapos háttértanulmány készítette elő [113]. A stratégiákban alapfeltételként szerepel az eszközfejlesztés, az informatikai eszközök használatára és új módszertanokra tanárképzés. Miközben a diákok az informatikai készségeket – most már – tantárgyi keretben, általános törvényszerűségeken keresztül sajátítják el a tanárok csak az IKT számukra szükséges aktuális elemeit tanulják. A moduláris, hiányokat pótló képzések miatt a szaktanároknak nincs átfogó képe arról, hogy a diákja

mit, mikor és miért tanul informatikából. Emiatt a digitálisan felkészült tanárok sem tudnak építeni a diákok informatika ismereteire. A VIII. mellékletben az stratégiák elemzése mellett, bemutatok néhány olyan témát, ahol az gátolja az informatika beépülését az oktatásba, hogy a tanár nem tanulta azt, amit a diákja.

Az elmúlt húsz év pedagógus-továbbképzési erőfeszítései és az elért eredmények azt mutatják, hogy az IKT-kompetenciák mennyiségi fejlesztése hosszú távon nem eredményes. Az MDOS [112] a pedagógusok IKER [114 p:5, 115 p:38] 4. szintű képzettségét várja el, ami lényegében azt jelenti, hogy minden pedagógusnak meg kellene tanulnia a közoktatási informatika tananyagot; az informatikai gondolkodás és a programozás ismeretek szükségesek nem csak a diákok, hanem minden pedagógus számára is. Azonban kollégáim körében azt tapasztalom, hogy ez az elvárás többségükben félelmet kelt. Az MDOS megjelenése óta az IKER 8-szintesre finomult [116], amelyből a pedagógusoknak sok szempontból a 8. szint ismerete lenne szükséges.

6 Az informatika oktatása

Mintegy negyven éve létezik informatikaoktatás, azonban tartalmában, formájában rengeteget változott ez idő alatt. Az új tudományterület születése, mibenlétének bizonytalanságai leképződtek az oktatásban is. Az egyetemi képzésben az ELTE-n a matematikus képzésből fejlődött a programtervező matematikus képzés, ahol az algoritmusokon volt a hangsúly, a BME-n villamosmérnökök képzésébe épült be a digitális technika, programozás. Az ELTE-n napjainkban még jellemző, hogy az Informatika karon valamilyen matematikus végzettsége is van az informatikát (programozást) oktatónak, a BME-n pedig még a kar is közös, így villamosmérnöki diplomával oktatnak informatikát. Az ELTE-n képeznek informatikatanárokat, így a tudásuk alapja a matematika, miközben a BME-n képezik a mérnöktanárokat pedagógussá, akár villamosmérnöki végzettségből. Mindeközben a közoktatásban a számítástechnika oktatása technikaórán az eszközök és programok használatának oktatását jelentette. Matematikaórán néhány évig választható volt a programozás modul, így matematikatanár tanította [72]. A tantárgyi oktatás megjelenésével a programozás kiszorult az általános képzéséből, csak emelt szinten (illetve szakmai képzésen) kellett tanítani, míg az általános képzésben az igényekhez igazodó eszközkezelés és alkalmazásoktatás volt jellemző. Az informatikaoktatásról szinte egyidőben két jellemzést hallhattam: „játszunk egész órán”, illetve „kódolni kell, ami senkinek sem megy, és nem is lesz rá szükségem”. Ezek egyikére sincs szükség, de mit és hogyan kell informatikából megtanítani, megtanulni?

A már fentebb vizsgált stratégiák [108–112] főleg kompetenciákat jelölnek meg, a tantervekben [65–68, 79, 80] a kompetenciák mellett a témakörök, altémák is rögzítve vannak. A kerettanterv [69, 70], illetve a helyi tantervek tartalmazzák a tananyagot mélységében. A szakmapolitikai döntéseket szakértői és tudományos elemzések kísérik: témakörök feltérképezése [86], tematizálása [117], elhelyezése a többi téma között [118], bevezetés akcióterve [119] és kivitelezésének elemzése [120]. A tapasztalatok nemzetközi megosztása eredményeként az informatikaoktatástól elvárt kompetencia fejlesztés területei és a tanítandó fő témakörök a digitálisan fejlett országokban azonosak, ennek a 2020-as NAT [68] szöveg szintjén megfelel.

Egy új tudományág oktatásának módszere is új, az alakuló tudomány oktatásának módszere is változik, alakul. Nem segíti az informatikaoktatást, hogy az oktatási módszerek gyakorlati alkalmazásában komoly szerepe van a hagyománynak: a tanár módszertani alapjait tipikusan az őt tanító tanár módszerei adják. Egy új tananyag esetén is, arra alapozza a tanár az oktatást,

hogy ő maga hogyan tanult. Az informatikaoktatás esetén az elmúlt negyven évben kimondottan rossz eredményt hozott, ha a tanár abból indult ki, hogy neki hogyan tanították vagy hogyan tanulta meg önmagától az egyes címszavakhoz társított tartalmakat, mert a tudomány és technika fejlődése nem csak a tartalmi, hanem tanítási-tanulási módszerben is megújulást igényelt. Az informatikaoktatás leminősítését eredményezte az is, hogy képzett tanár hiányában OKJ vizsgával is lehetett informatikát tanítani a közoktatásban, sőt, napjainkban is tanítanak „demonstrátorok" és szakemberek, akik, bár tudják, hogy mit kell tanítani, a saját élményükön túlmenően nem tudják, hogy hogyan.

Néhány hibás, nem hatékony módszer az informatika egyes területeinek oktatására, ami jellemző volt (és még ma is találkozhatunk vele):

- algoritmizálás oktatását a teafőzés algoritmusával kezdeni (bővebben [121]);
- programozás oktatását a „Hello World" kiírásával kezdeni;
- tanórát az egér vagy billentyűzet használatának gyakorlásával (játékkal) kezdeni;
- tanórát egér vagy billentyűzet használatának gyakorlásával (játékkal) befejezni;
- alkalmazás felhasználói dokumentációját (menüt) tanítani, tankönyvként használni a kézikönyvet;
- algoritmus implementációt betanítani (tételeket magoltatni) …

Az informatikaoktatás megszületését és fejlődését saját problémáin túl a külső elvárások is jelentősen megnehezítik. Jellemzően olyanok alkottak ítélet, fogalmaznak meg elvárásokat, akik nem értik a problémákat, ők maguk sohasem próbálták megtanulni azt, amiről véleményt alkotnak. Például:

- nem tanult informatikát, nem tanult programozást, és állítja, hogy a gyermeke sem fogja tudni megtanulni a programozást, felesleges megpróbálnia, ezzel időt pazarolni;
- nem tanult programozást, de úgy gondolja, hogy az kell; az a véleménye, hogy a mai gyerek okos és már mobillal fekszik, kel… egy óra alatt a programozást is meg tudja tanulni, csak meg kellene neki mutatni;
- informatika csak a mobil telefon használatát jelenti, ezért úgy gondolja, hogy a gépelés felesleges ismeret (van, aki szerint ma már a kézírás is felesleges);
- bár önmaga nem ért hozzá, de a gyermeke mindent tud, a tanár hibája, hogy nem kitűnő informatikából;
- a gyerek nagyon érdeklődik az informatika iránt (egésznap játszik a számítógépen), tehát tehetséges informatikából …

A fentieknek részben következménye, részben erősítő tényező, hogy a tanuló személy sem képes saját tudását értékelni. A tanuló is érezheti magát „hülyének” a programozáshoz és „profinak” az egyes alkalmazásokban úgy, hogy ezeknek a minősítéseknek csekély köze van a valósághoz. Egy konkrét példa: Állásinterjún derül ki, hogy az „Excelből haladó ismeretekkel rendelkező” haladó szintje az egyszerű statisztikai függvény használata; reménybeli munkakörében leggyakrabban használt FKERES() függvény paraméterezését nem sikerült egy órán belül megoldania. A jelenséget nemzetközi kutatás is kimutatta: Az ECDL vizsgarendszer vizsgázóinak önértékelése és vizsgaeredménye között rendkívül nagy az eltérés. A Perception & Reality [122] felmérés szerint utolsó éves középiskolások és egyetemisták körében a szövegszerkesztés, a táblázatkezelés és a prezentáció témakörökben az átlagosan 78%-ra önértékelt tudás mért értéke csak 53% lett.

Szülőkkel, kollégákkal, diákokkal beszélgetve, úton-útfélen hallgatva az informatika oktatásáról alkotott véleményeket fogalmazódott meg bennem, hogy az informatika oktatása nem jelentheti a hétköznapi aktuális problémákból összelapátolt digitális kultrutymót. Ezért

- elemeztem az oktatás pedagógiai hátterét a 3. fejezetben;
- meghatároztam, hogy mit jelent az „Informatika”, mint tudományág a 4. fejezetben és
- az 5. fejezetben elemeztem, hogy hogyan jelenik meg az informatika az oktatásban.

Egy hagyományos tantárgy, egy régóta elismert tudomány esetén ezekre a meghatározásokra elég egy-egy hivatkozás. A kutatásom során a meghatározások az oktatás szempontjából alapvető kérdésekre adott válaszok. A válaszok megfelelnek az informatikaoktatásra vonatkozó szakmai elvárásoknak [86, 117–120]. Az informatikaoktatásra vonatkozó megállapításaimban ezekre a válaszokra hivatkozom. Akkor is, amikor megfogalmazom az informatikaoktatás mibenlétét, amihez a módszerekkel kapcsolatos kérdéseket majd felvetem:

Az informatikaoktatás keretében az informatikatudomány alapjait kell tanítani.[13]

Azaz az informatikaoktatás azt jelenti, hogy

- az informatikatudomány (4.7 specifikációnak megfelelően) kompetenciáit, világértelmezési módjait;
- szakmailag és módszertanilag képzett informatikatanár tanítja [85].

Az informatika tudomány meghatározásában a tudomány fejlődésének dinamikájából következik, hogy az informatikaoktatás a folyamatosan változó aktuális problémák megoldásait kell, hogy integrálja, képes kell legyen arra, hogy a ma fiataljait felkészítse egy ma még ismeretlen

[13] A T1 és T2 tézisekhez az oktatás tárgyának (és céljának) a meghatározása.

jövőbeli helyzetben a problémamegoldásra, élethosszig tanulásra. Az informatikaoktatás legfontosabb feladata az informatikai (számítógépes) gondolkodás fejlesztése.

6.1 Az Informatikaoktatás tartalmi koncepciója

Az informatikaoktatás tartalmi meghatározásakor nagyon gyakori, hogy egy-egy szoftver vagy eszköz használatának tanítását nevezik meg. Eszköz esetén a kezelési útmutató, szoftver esetén a szoftver dokumentációja jelenti a tananyagot. Ezt a tartalmi meghatározást szokás a „menü megtanítása" néven is emlegetni. Informatikatanárok jelentős részének egyértelmű, hogy nem ez a feladata az informatikaoktatásnak, de számos képzés csak erről szól, a tankönyvek jelentős részére is ez a feldolgozási mód jellemző és az oktatóvideók is többnyire ebbe a kategóriába tartoznak. Tény, hogy az eszközök, szoftverek használati módjának ismerete alapvető – az adott felhasználói ismeret LAU^{L}1–5a szintje, de az oktatásnak nem a konkrét eszköz megtanítására kell koncentrálnia, hanem arra, hogy hogyan tanuljuk meg önállóan, az elérhető dokumentációk alapján kezelni az eszközt, használni a szoftvert. Ennek elsajátításában a menü megtanítása a ^{L}1–2 fázis.

A tantervek elvileg eszköz- és szoftverfüggetlenek. Ezekben az informatikaoktatás tartalmi meghatározásakor az eszköztípus, szoftvertípus felhasználási módja kap hangsúlyt. Tantervek alapján azt tanítjuk, hogy mire használhatók az egyes szoftverek, a beépített eszközök. A tanítás lényege, hogy a diákok elsajátítsák, hogyan használják az adott eszközöket egyes problémák megoldására. Ez az oktatás felkészíti a tanulókat jövőbeli ismert helyzetekben a feladatok megoldására – LAU^{L}5b fázis. A tanítás eredménye, hogy a tanuló az eszköz- és szoftverhasználatból egyes területeken rutint szerez. Komoly problémát jelent azonban, hogy a feladatok változására, új problémák megoldására, kreativitásra, innovációra nem készít fel. A hiányos tudás néhány tipikus jele: az elsajátított tudás gyorsan elavul; egyes feladatok túl nőnek a szoftver képességein; a megoldási idő hosszabb lesz, mint ha fejben/papíron oldanák meg a feladatot, azaz a választott megoldás nem hatékony; a megoldás esetleg hibát tartalmaz, aminek az ellenőrzése nem hatékony.

Elemezve a tanítást és tananyagot megállapítható, hogy az eszközök és alkalmazások felhasználásának oktatása nem felel meg az informatikaoktatás meghatározásánál leírtaknak. *Pusztán az egyes szoftverek felhasználási módjának tanítása – bár szükséges – nem informatikaoktatás, nem az informatikatudomány oktatása.*

Az informatikaoktatás[14]:

[14] A T1, T2 tézisekben megfogalmazott informatikaoktatás értelmezése.

- Az egyes eszközök, szoftverek, illetve ezek működésének **modellezés**ét jelenti.
- Minden témánál az **adat absztrakció**ját, az elvárt **algoritmus**okat és az adott eszközre, szoftverbe írt **algoritmus**okat kell a tanítás fókuszába helyezni.
- Minden témánál fel kell tenni a kérdést, hogy a felhasznált eszközt és szoftvert milyen használatra **tervez**ték az alkotói, a tervezés és megvalósítás során milyen **döntés**eket hoztak, mit tekintettek **alapértelmezett**nek és mennyire **paraméterezhető** az eszköz, illetve szoftver működése.

A kérdések felvetése a lényeges, a válaszok korosztályonként eltérő mélységűek és egyébként is nagyon sokfélék lehetnek.

- A válaszokból kell következnie a **probléma megoldási mód**jának.
- Minden informatikai probléma megoldásánál alapvető minőségi szempont a megoldás **teljessége**, **hatékonysága**, újra **felhasználható**sága, **rendszerbe illeszthető**sége.

A programozás az informatika gépre történő **implementáció**ja; a szoftver a programozásnak, azaz egy implementációnak eredménye. Ezért *az informatikaoktatás tartalma jelentős részben a programozásról szól, akkor is, amikor kész szoftverek vagy eszközök működését tanulmányozzuk.*

A jövőben szükséges, a változásokat követni képes kompetenciák:

- az eszközök, szoftverek elvárt működése megfogalmazásának képessége;
- az elvárásokat teljesítő eszköz, szoftver kiválasztásának képessége;
- az adott eszköz, szoftver konkrét paraméterezhetősége megismerésének a képessége;
- az eszköz, szoftver használatában – adott, gyakran előforduló feladatra – a rutin megszerzésének képessége (felhasználói alkalmazás L5b);
- az eszköz, szoftver (nem rutin jellegű) komplex feladat megoldásába integrálásának képessége.

A kutatásom során az informatika témaköreinek oktatását ennek megfelelően gondoltam újra. Az eszköz- és szoftverhasználat oktatása helyett az informatikát helyeztem előtérbe. Jellemzően ezt úgy valósítottam meg, hogy a programozáshoz szükséges alapokat, fogalmakat, részképességeket tanítottam minden tananyagban, ezzel mindennapivá tettem az *informatikai gondolkodás* gyakorlását, fejlesztését, az informatikai szemléletmódot. Feltűnő – később részletesebben vizsgálom –, hogy az informatikai gondolkodás fejlesztéséhez egy-egy tanagyag L5c fázisa kapcsolható. A fenti felsorolásban csak egy kivétel – rutin – található.

Az informatikaoktatás fenti definíciója megvalósítja Wing javaslatát, az informatikai gondolkodás középpontba helyezését [123], feloldja az alkalmazás kontra programozás dilemmát

[124], mivel az alkalmazásban az algoritmust [125] a programozásban az alkalmazási lehetőségeket [126] is tartalmazza.

6.2 Az informatikaoktatás tartalma[15]

A kutatás során 2009-es és 2013-as NAT szerint képzett csoportokat tanítottam, de a jövőbeli használhatóság érdekében a 2020-as NAT Digitális Kultúra tantárgyában megfogalmazott fejlesztési területekhez és eredménycélokhoz viszonyítva mutatom be, hogy mit jelent – a fentebb leírt értelemben – informatikát oktatni. A részletes leírás meghaladja e dokumentum kereteit. Egy vázlatos elemző (de még így is hosszú) leírás olvasható a IX. mellékletben, amiből a legfontosabb gondolatokat és következtetéseket emelem ki itt.

Az egyes NAT-ok és a 2020-as NAT között a legnagyobb eltérés az egyes témák súlyozásában van. Ahogy azt a NAT (és kerettanterv) készítői szakmai rendezvényeken elmondták, illetve az ÉGIG[16] kezdeményezésére szervezett informatika és technika munkaközösségi találkozó jegyzőkönyvében [127] szerepel „A három fő témakör aránya a jelenlegi magyar gyakorlatban: 80%–10%–10%, míg a kívánatos az 50%–30%–20% vagy 40%–40%–20% lenne (iskolatípustól függően).” Az első szám a digitális írástudás, a második a problémamegoldás, a harmadik az információs technológiák témakör súlya, míg az eszközök használata „beépül” a többi témakörbe. Zsakó László 2015-ben a 2020-as NAT-ra vonatkozóan ajánlásában [72] – bár némiképp eltérő megnevezésekkel – ugyanezeket az arányokat adta meg.

Mivel a NAT-ot felmenő rendszerben vezetik be, a vizsgált, esetenként hatévfolyamos képzésben résztvevő diákok közül 2018-ban ment ki az utolsó 2009-es NAT szerint tanuló csoport, miközben a négyévfolyamos képzésű csoportokból 2017-ben végzett az első 2013-as NAT szerinti társaság, de helyi tantervtől függően 1-2 évvel korábban fejeződött be az informatikaoktatásuk. A tantervi változásoktól függetlenül, amikor 1-1 téma oktatására lehetőségem volt, az itt bemutatott „új” tartalmakat a későbbiekben részletezett módszerekkel tanítottam.

Az informatika oktatása során számomra (4.7 alapján) vezető szempontok voltak, hogy

- az informatika fejlődik, változik, újraértelmezi a fogalmakat;
- az oktatás jellemző célja az adatok értelmezésének és a modellezésnek a tanítása, az informatikai gondolkodás fejlesztése.

[15] Ebben a fejezetben a T2 tézisből az egyes témák informatikatudományi megközelítését mutatom be.
[16] Élenjáró Gimnáziumok Igazgatóinak Grémiuma

Ezért jellemzően nem definíciók, hanem kérdések, problémafelvetések sorával jellemezhető az informatikaoktatás. A feltett kérdések, problémák, értelmezési feladatok nem feltétlenül köthetők korosztályhoz, a lehetséges válaszok azonban jellemzők lehetnek egy-egy korosztályra, a válaszoló tudásprofiljára, tapasztalatára.

6.2.1 Az informatikai eszközök használata (1.)

Látható, hogy a „fejlesztési területek” az eszközhasználatra vonatkoznak, az „eredménycél” pedig egy gyakorlati tevékenységre való képesség. Ezek nem biztosítják sem a jövőbeni önálló fejlődést, sem az informatikai gondolkodás fejlesztését. A megnevezett eredménycélokon túlmutató hosszútávú célok eléréséhez, az informatika tudomány szemléletmódját figyelembe véve a téma tanításakor a főbb fejlesztési területek (részletesen: IX/1. melléklet):

- A használt eszközök és alkalmazások működésének *modellezése*.
- Az eszközökre jellemző *adatok értelmezése*.
- Az *adat* különböző *megjelenési*, *tárolás*i és *továbbítás*i módjainak megismerése.
- Az eszközök és alkalmazások működési *algoritmus*ainak tanulmányozása, *modellezés*e.
- *Informatikai gondolkodás* fejlesztése, adott feladatokra és problémákra *hatékony* megoldások *tervezés*e és *kivitelezés*e, a megoldások *verifikáció*ja és *validáció*ja.
- Működési problémák beazonosítása a modellek és a jelenség alapján; hibás megoldások, hibalehetőségek felderítése.

Az Informatikai eszközök használata a legkisebb témakör, amely a többi témakörbe integrálva tanítandó. Óraszám nem tartozik hozzá, ezért az oktatás során külön figyelmet kell fordítani az informatikai kérdések felvetésére, az ismeretek elsajátítása során az informatikai gondolkodás fejlesztésére. A NAT eredménycéljaiban olvasható „használ” és „választ” kiegészítéseként az informatika tudáshoz a „modellez”, „tanulmányoz”, „tervez”, „felderít” cselekvések vezetnek.

6.2.2 Információs technológiák (4.)

Jellemző eredménycél, hogy a diák az adott témát „ismeri” és lehetőséget helyesen „használja”. Az eredménycélok az éppen aktuális információs technológia alkalmazására irányulnak (IX/2. melléklet). Az informatikai tudás ezen felül azt jelenti, hogy az egyén érti a szerepét az információs társadalomban, tisztában van azzal, hogy *rendszerkomponense*ket használ, amivel ő is a rendszer egy komponensévé válik. Érti a használt technológiák *működési elv*ét, helyesen *értelmezi* a kapott *adatokat*, átlátja, hogy az egyes technológiák használata során milyen adatokat szolgáltat önmagáról, fel tudja mérni, hogy ennek milyen hatása lehet másokra, illetve önmagára nézve.

6.2.3 Digitális írástudás (2.)

A digitális írástudás – a NAT témakörök alapján olyan készség, amely – a gondolataink, az érzéseink kifejezésére ad lehetőséget. A „digitális írás” során digitálisan tárolt dokumentumot hozunk létre, amelynek olvasója, *értelmező*je – alapértelmezetten – humán egyed. A dokumentumok eszerint az ember-ember közötti kommunikáció csatornái. Ezt alapul véve a IX/3. mellékletben részletesen elemezve határozom meg a témakör oktatásának a tartalmát.

A digitális írás a gondolat *kódolás*a. Az egyes dokumentumtípusok használatának ismerete olyan, mint egy adott nyelven a beszéd képessége, amely azonban a nyelvtan és beszédstílus, esetenként a metakommunikáció ismerete és használata nélkül zajos, pontatlan kommunikációt eredményez.

A digitális írástudásnak is hangsúlyos témája a szöveg rögzítése. Azonban a beszélt nyelv és írás soha nem tudja önmagában pontosan kifejezni a gondolatot, érzést, állapotot. A pontosítást például metakommunikációval, intonációval érhetjük el. A digitális írás során a szöveg karakterek képeként és karaktersorozatként is értelmezhető, amelynek információtartalmát kiegészíti a szöveg digitalizálásának módja, a karakterek elrendezésének módja (például a szöveg tördelése, kapcsolása), a vizuális megjelenést befolyásoló tulajdonságok és a hozzáadott nem szöveges – de jellemzően vizuális – elemek.

Példaként a digitális írás módjaira és ezek kommunikációs szerepére nézzük Apollinaire versét, amelynek szövegét Radnóti Miklós fordította.

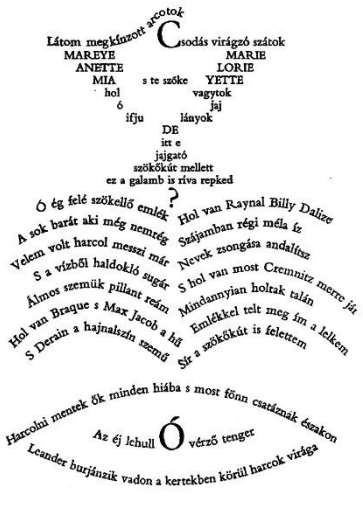
- A szöveg szavalható, ami digitálisan rögzíthető. A tartalom alapján a *hang*dokumentumhoz háttérzaj (vagy zene) adható.
- A mű eredetileg képvers, digitalizált formája *rasztergrafikus kép*, a vers szövege is képi elemként jelenik meg. (6. ábra)
- A szöveget *karaktersorozat*ként tárolva, a tördelés és *tipográfia* elemeit használhatjuk további információ közvetítésre. (7. ábra)
- A szöveget kifejezésekre, szavakra szedve *vektorgrafikus* algoritmusokat használva szófelhőt készíthetünk belőle. (8. ábra)

  Bár a szöveg (magyar fordítása) adott, az interpretációnak számos eszköze van, amelylyel többletinformációt szolgáltatunk. A vers címe a képi megjelenésre utal, a szöveges tartalom a látvány és a közlő belső világa közötti kapcsolatot fejezi ki.

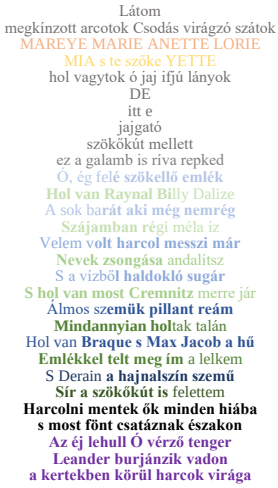
A példa illusztrálja, hogy digitális írástudás nem pusztán a szöveg bevitele, hanem a többlet információnak a szándéknak megfelelő, minél pontosabb, *hatékony* hozzáadása. A digitális írás lényege, hogy a dokumentum *értelmezés*e könnyű legyen és az eredmény a közlés szándékának

megfeleljen. Az emberi „vétel” eszközei: látás, hallás, tapintás, ízlelés, szaglás. amiből több mint 80%-os dominanciája van a látásnak [128 p:14]

1. táblázat: Apollinaire – Radnóti Miklós: A megsebzett galamb és a szökőkút implementációi



6. ábra: képvers[17]



7. ábra: formázott szöveg

8. ábra: generált[18] *szófelhő*

A témakör oktatása/tanulása során megvizsgálandó, hogy az adott médiaelemek a képi, szöveges és hang *adatok rögzítés*ére milyen *absztrakció*t használnak, az adott absztrakció milyen *tulajdonság*okkal rendelkezik, mennyire felel meg a közlési szándéknak és hogyan segíti az *értelmezés*t [129].

Több fejlesztendő terület említi a multimédiát, ahol a „multi” a kép, hang, videó kombinációt jelenti. A fenti példa egy szöveg mint a gondolat nyelvi kifejezése, de digitális megjelenése számos formátumban lehetséges, így a multimédia minden változatában *implementál*ható, *kódol*ható.

A NAT-ban a digitális írástudás témakör látványosan megáll az alkalmazástípusok tanítása szintjén. Emiatt kérdéses, hogy miért kellene informatikából egyik vagy másik alkalmazástípust (weblapkészítés, videókészítés) tanítani, ami az informatikaoktatás devalválódását eredményezi. A digitális írástudás a digitális eszközök – alkalmazások, appok – *hatékony felhasználás*a érdekében szükséges, ehhez *informatikai gondolkodás*ra van szükség. A digitális írástudás szoftverek *célszerű használat*át jelenti. A szoftverek az informatikatudomány termékei, ebből következően a működésüket informatikai szempontok figyelembevételével kell vizsgálni. A digitális írástudás témakörben feltüntetett alkalmazástípusok felhasználói ismeretén túl szükséges

[17] A képvers és szöveg forrása: https://hu.wikisource.org/wiki/A_megsebzett_galamb_és_a_szökőkút [2021.10.30]
[18] Készült a szöveg felhasználásával https://wordart.com/ használatával. [2021.10.30]

az alkalmazások működésének a modellezése és a *modell*ből következő felhasználói elvek ismerete.

Az aktuálisan tanított alkalmazástípusok felsorolásán túllépve, a témakör informatikai tartalmához az egyes médiákat (karaktersorozat, hang, pixelgrafikus kép, vektorgrafikus kép…) külön-külön kell jellemezni és értelmezni kell a különböző multimédiás dokumentumokon belül a szerepüket. Az informatikai megközelítést az *adatok*kal és *modellek*kel operáló *absztrakció* jelenti. Ezt követően lehet egy-egy cél érdekében ezen elemekből többfélét kombinálni egy multimédiás dokumentumban. A multimédiás alkalmazások célszerű felhasználásához ismerni kell az egyes elemek *objektummodell*jét és az objektumok kombinálási lehetőségeit.

Ezért tartom fontosnak például a Paint mint egyszerű, tisztán rasztergrafikuskép készítő alkalmazás tanítását. Ezen az alkalmazáson keresztül elemeztem, hogy hogyan lehet a menü és az alkalmazás gyakorlati trükkjei mellett a rasztergrafikuskép *adatmodell*jét is tanítani, az informatikai ismeretek mentén akár a kép programozással történő módosítását is bemutatni [130]. A multimédiás alkalmazások tanítása előtt vagy annak kezdeti szakaszában tanítom a többi felhasználni kívánt médiaelemet is, így a karakterek *kódolás*át, nyelvtani és helyesírási szabályokat is.

A multimédiás tartalmat az oktatás során érdemes a vétel (dekódolás) alapján is informatikai szempontok mentén megvizsgálni, mert ez a választott médium célszerűségének meghatározó szempontja.

- Melyek azok, amelyek a lineáris adatsor közvetítésére alkalmasak (pl.: hang);
- melyek engednek a feldolgozásra több bejárási utat (pl.: weblap) és
- melyik médiák adnak a feldolgozás, a dekódolás oldalán szabadságot (pl.: kép esetén a szemlélő fókuszál, talál meg egy-egy részletet).

A médiákat az ember *dekódol*ja. Ennek megfelelően az oktatásban ki kellene térni arra, hogy már *kódolás*kor figyelemmel kell lenni a dekódoló képességeire[19].

- A vizuális tartalmak értelmezése könnyebb (gyorsabb), mint az audióé [128, 129], mert az audió adás idő dimenziója kényszert jelent a feldolgozási sebességre, gátolja az ismétlést és visszatekintést. (Például előadás megfelelő sebességgel, gondolattérkép értelmezése)

[19] Az érzékszervek biológiai adottságból származó jelfeldolgozási képessége.

- A hang egyedi dekódolása (füldugóval) a közvetlen környezet érzékelését gyengíti, a kakofóniából a szükséges hangok kiszűrése fárasztó és nehézkes. (Például hangos prezentáció készítése tanteremben.)
- A 3D ábrázolás realisztikusabb, mint a 2D de – főleg mozgó változata – az embert érő vizuális és mozgásérzékelési ingerek ellentmondásos jelzései miatt rosszullétet (tengeri betegség) okozhatnak. (Például játékok, Prezi animációk.)

A multimédiás tartalmak készítésekor a *hatékonyság*ot több szempontból kell vizsgálni:

- Az elkészítés hatékonysága a közlési szándék és az elkészítési idő függvényében értelmezhető.
- A dokumentum tárolásának hatékonysága méretbeli és adatbiztonsági kérdéseket vet fel.
- A felhasználás hatékonysága a közvetített információ megszerzésének lehetősége, ideje, teljessége alapján jellemezhető.

A digitális írástudás nem csak azt jelenti, hogy ismerünk és célszerűen fel tudunk használni egy tucatnyi alkalmazást gondolataink, adataink közlésére, hanem azt is, hogy az alkalmazásoknak

- ismerjük a *működési elv*ét: az *adat*ait (*objektum*ait) és *algoritmus*ait;
- értjük az alkalmazások működése eltéréseinek az okát és
- ennek ismeretében *választ*juk meg a közlés, tárolás és feldolgozás szempontjából legmegfelelőbb alkalmazást, multimédia elemeket és *megoldási módszerek*et.

A digitális írástudás képessé tesz arra, hogy új alkalmazások megjelenésekor annak működési elvét – a korábban megismertekkel összehasonlítva – *önállóan* felderítsük, *hatékony* használatát elsajátítsuk.

6.2.4 Problémamegoldás informatikai eszközökkel és módszerekkel (3.)

A téma címe alapján a tantárgyon belül itt oktatunk informatikát. Valójában a témakörön belül az ember-számítógép kommunikáción van a hangsúly. Középpontban a gép vezérlése, a gépen tárolt adatok feldolgozása áll. Informatika szempontjából, a programozás különböző módszereinek az alapjai szerepelnek ebben a témában:

- imperatívan programozás;
- deklaratívan adatbázis-kezelés;
- az objektum orientált és esemény-vezérelt programozáshoz a robotika és a blokknyelven programozás nyújt alapot;
- a táblázatkezelők használata a funkcionális programozás gondolkodásmódját igényli, adatkezelése, megoldásai a programozást több szempontból modellezik. [131]

Ezért a programozással kapcsolatos ismeretek az imperatív nyelv tanítása előtt, például a táblázatkezelés oktatása során előkészíthetők [132]. Erre mutattam néhány példát a How to Teach Programming Indirectly – Using Spreadsheet Application [133] cikkemben. Ebben és más módszertani elemzésekben is [134] a ciklusképzés képletek másolásával valósul meg. A tömbfüggvények használatával képletbe is foglalhatjuk a műveletek ciklikus elvégzését, ezzel a SPREGO [135] gondolkodási módszert kapjuk, ami a táblázatkezelő eszközeit programozási nyelvként használja. Figyelemre méltó, hogy az MS 365 „Excel Formula Language" már teljes funkcionális nyelv: a rendezés is függvény, az eredmény dinamikus tömb, LAMBDA kifejezésként egyéni függvényt (rekurzívat is) lehet írni.

Az adatbázis-kezelő alkalmazások a táblázatkezelőknél is jobban hasonlítanak a programok fejlesztő környezetéhez. Két nyelv, az adatdefiníciós és a lekérdezőnyelv kódjai futnak a kattintgatásokkal vagy szövegesen kiadott utasítások hátterében. Az SQL nyelv ismerete, csak emelt szinten elvárt, a feladatokat lekérdező-rácson (QBE felületen) megoldva is, a blokknyelvekhez hasonlóan programozásról beszélhetünk.

Az informatikaoktatás tartalma szempontjából – a IX/4. mellékletben részletezve –vizsgálom:

- a témakörön belül az egyes programozással kapcsolatos fogalmak hogyan jelennek meg, hogyan fejlődnek.
- a többi témakörhöz van-e kapcsolat.

6.2.4.1 Adat

Az adat absztrakciója az informatika tanulásának kezdetén elnagyolt, de a digitális írástudás minden alkalmazásánál felvetendő, így passzív vagy mellékes ismeretként korán megjelenik (LAU ᴸ1P vagy ᴸ1M). A táblázatkezelés tanulása során válik aktívvá az elemi adattípusok ismerete, különül el a szöveg, szám (érték), a logikai típus és a hivatkozás.

A szám típuson belül az egészek mellett a tapasztalás mutat rá a nem egész számok véges jegyű digitális ábrázolásának problémájára [133]. A dátum/idő kezelésének egyik problémája a „végtelensége". Bár a mai iskolások nem élték meg az Y2K problémát, a történelmi tanulmányaik során tanult dátumok jelentős részét szöveges adatként kezelik a szoftverek, miközben a jelen eseményeinek dátumát, idejét, az eltelt időt szám(érték) típusú adat jellemzi. A dátum/idő kezelés során tapasztalható, hogy a digitálisan tárolt érték esetén nem csak a végtelen nagy, hanem a végtelenül kicsi sem értelmezhető. Az egység lehet a nap vagy a másodperc és ennek lehet törtrészével is számolni, de az idő ábrázolása pontosságát az adat tárolási mérete korlátozza. A logikai kifejezések IGAZ/HAMIS értéke a megfelelő statisztikai függvények számára

0 és 1, más értelmezéssel 0 és –1. Ugyanakkor a számérték logikai kifejezésként értelmezhető, ahol a 0 a HAMIS kifejezése, minden más (pozitív, negatív) érték típusú adat IGAZ. Az adatkezelés kezdetén elkülönül az érték típus a szöveg típustól. A mennyiség mértékegységét csak formátumként lehet megadni, a szöveges típussal a szoftverek nem tudnak számolni és logikai értéke sincs.

A szöveg mint adattípus kezelése során intuitíven ismertté válik a sorozat, karaktersorozat. Ennél sokszor korábban, a pixelgrafikus ábrázolással kezd kialakulni a tömb fogalma [128], ami azután táblázatkezelésben a cellákra hivatkozással folytatódik. A matematikatanulmányok előtt évekkel, alkalmazás szintjén ismert lesz az index, a sor és oszlop azonosító (koordináta). Gyakorlati alkalmazásként jelenik meg az adat helyett az adatra való hivatkozás.

Elsősorban a grafika, animáció, blokknyelvek révén alakul az objektum fogalma, de a szövegszerkesztés és táblázatkezelés – valamint minden más alkalmazás – oktatása során ki kell térni az alkalmazás objektumainak a jellemzésére. Az objektum egyszerűsített adatleírása a rekord (struktúra). A programozás tanításakor a többi témakörben korábban szerzett tapasztalatokra kell alapozni, a különböző adattípusok és az osztály, illetve az objektum fogalmát a programozás tanításakor kreatívan használjuk: a korábban már használt dolgokat – adatokat és objektumokat – tudatosan létrehozunk, módosítunk, törlünk.

Az alkalmazások tanulmányozása vezeti fel az adatelérésének kérdéseit is. A háttértáron tárolt adatra hivatkozás fájlnévvel, amit megnyitva a memóriába másolat töltődik be – felsős tananyag. Az adatról (képfájlról) készült másolat kerül a szöveges dokumentumba – felsős tananyag, de weblapban abszolút vagy relatív hivatkozással adjuk meg középiskolában. A beágyazott, illetve a csatolt multimédia, az alapértelmezett tulajdonságok módosítása dokumentumban kódolt vagy csatolt stílusmeghatározásokkal mind az adat közvetlen majd paraméteres elérését példázza. Ezt egészítik ki a táblázatkezelésből tanult függvények, ebben argumentumként a hivatkozások használata. Az egyes cellák hivatkozásához hozzárendelt érték újabb absztrakciójaként tananyag – az INDEX() függvénnyel – a hivatkozással megadott tömbön belül, relatívan megadott helyhez tartozó adat elérése. Többszörös kapcsolaton keresztül történő adatelérést gyakorol már az általános iskolás is az adatok fa struktúrájú mapparendszerben történő navigálással, ezt követi középiskolában adatbáziskezelésből a többtáblás relációs adatbázisokban való eligazodás.

Kulcsfontosságú a kulcs fogalma, amely egy adatstruktúra összekapcsolt adatait azonosítja, egyben hivatkozási eszköz. Egymásután rögzített adatok, adatstruktúrák használatában az index

fogalma kerül középpontba, megtámogatva a matematikai analízissel, ahol zérushely, szélsőértékhely megadása jelenti a jellemzés alapját, táblázatkezelésben és programozásban is a HOL.VAN() és a tömbbéli index meghatározása a típusfeladat, mert ebből akár az adott helyen lévő objektum bármely adata, akár a szomszédok hatékonyan, címszámítással, hivatkozással elérhetők.

Az egyes alkalmazások tanulása és működésének elemzése során kialakult, megtapasztalt fogalmak a programozás tanulása során jól hasznosíthatók.

6.2.4.2 Algoritmus, értelmezés, számítás

A hagyományos közoktatási programozásoktatás az algoritmusokra fókuszál, ezzel együtt a módszeres, procedurális, imperatív programozás jelenti a tananyagot. Az OOP, a robotika fejlődése, a grafikus megjelenítés érdekesebbé tette a programozást, egyre könnyebb működő programot készíteni, de ez nem igényli algoritmusok kigondolását, a hagyományos értelemben vett módszeres programozást. Az 4.4 (Adat és algoritmus) fejezetben láthattuk, hogy az OOP kiszorítja az algoritmust. Az OOP lényege, hogy a programban szereplő objektumok önmagukon belülről vezérelve változtatják állapotukat, amint az látható a VI/2. mellékletben olvasható Eratosztenészi szita példa OOP megoldásán. Ebből következik, hogy előbb lesznek egy programban összetett objektumok, és csak utána algoritmusok [60].

A programozás NAT szerinti bevezetése a blokkalapú programozási nyelvvel OOP alapú, lehetőséget ad az állapotok változásának közvetlen vezérlésére, de egy animáció – prezentáció – elkészítésénél nem vár el többet.

Hagyományosan az algoritmizálás oktatását a „teafőzés” vagy hasonló hétköznapi algoritmussal vezetik be, amiről bemutattam [121], hogy az algoritmusban használt „adat” pontatlan definíciója miatt nehezebb az adat és algoritmus fogalmának, illetve ezek kapcsolatának megértése. A tantervben szereplő padlórobotok és a blokknyelvű programozás, a fizikailag megépíthető (például LEGO) robotok ezt a problémát kiküszöbölik, mert létükkel az objektum (az adat) definíciója is egyértelmű lesz. Az algoritmusok tanításához mintaként tekinthetjük egy létező (vagy elkészíthető) robot, illetve objektum tulajdonságait, metódusait.

Az algoritmizálás tanítását a középiskolában a NAT szerint „típusalgoritmusok”, illetve „egyszerű algoritmusok” tanítása követi. Ezek az egyszerű (elemi) sorozathoz értéket rendelő programozási tételek (algoritmusminták), tanításukhoz az OOP programok továbbfejlesztése helyett inkább a táblázatkezelés, adatbáziskezelés ismeretekre érdemes építeni. A gyakorlatban, alkalmazásokban található megoldások megfigyelése, „utánzása” több és célravezetőbb lehetőséget ad a megértésre, begyakorlásra. Az „utánzás” során tisztázni kell a humán természetes és

a gépi megoldás közötti eltéréseket. A cserénél nincs „levegőben” adat. A gép nem tud 2 vagy 3 dimenzióban szkennelve keresni. A legnagyobbat lehet „szintjelző metszetével” megadni. Az összegzés kumuláció, nem jó helyiértékenként végezni a többtagú összeadást.

Az algoritmusok tanítását jóval a programozás tanítása előtt (akár alsó tagozaton) el lehet kezdeni, de ki kell rá térni a táblázatkezelés megfelelő függvényeinek tanítása során is, a függvény működésének értelmezésével. Így a tipikus algoritmusok a digitális írástudás oktatásának jellemző pontjain taníthatók:

- Csere algoritmusát nem a rendezéssel, hanem jóval korábban, a rasztergrafikában képrészletek cseréje; szövegben formátumok vagy szövegrészletek cseréjével, táblázatban cellák, sorok, oszlopok felcserélésével érdemes tanítani.
- Keresés, eldöntés, kiválasztás többféle adatstruktúrában feladat. Fában keresést fájlkezelő modellezésével lehet algoritmizálni. Sorozatban keresés lineárisan és logaritmikusan a legfontosabb tanítandó algoritmusok, amit a táblázatkezelés keresőfüggvényeinek működése során is kell tanítani. A lineáris keresés kódja többféle gondolkodásmódot tükrözhet, ezt mutatom be a VI/1 mellékletben, cikkekben [57, 58] is elemzem és diákoknak is minden releváns témánál tanítom. A különböző megoldások tanórai elemzése hasznos, mivel a tanulók így gyakorolják a „kód olvasást”, társaik kódjának megértését, az alternatív gondolatmenetek követését. Az ember másképp csinálja: szkennelve keres, 2 vagy 3 dimenzióban.
- A szélsőérték-keresés algoritmusát „ki lehet találni” – és lehet animációval prezentálni – akár táblázatkezelés, akár adatbázis-kezelés MAX() függvényének tanításakor, de a hangsúly sokkal inkább azon van, hogy jellemzően a legnagyobb érték mellett/helyett annak helye vagy másik kapcsolódó adat a válasz. Emiatt a feladat táblázatkezelőben és adatbáziskezelőben összetett. Az összetettsége egyben azt is jelenti, hogy a végrehajtás nem hatékony, lehet *hatékony*abb algoritmust és programot írni rá.
- Felső tagozaton táblázatkezelésben elvárt az egyszerű statisztikai függvények használata: a megszámlálás, az összegzés, az átlag- és a szélsőértékfüggvényeket jelenti. Már ekkor tisztázni kell a megszámlálás és összegzés közötti különbséget, ami az algoritmusok tanítását is jelenti. Tisztázni kell, hogy mikor jó megoldás néhány cella összegét megadni, melyek azok az esetek, amikor a tartomány celláira (még, ha csak 1 vagy 2 cellára akkor is) végzett összegzés a helyes megoldás.
- A statisztikai függvények tanításában meg lehet fogalmazni a „nagy összegzés tétel” algoritmusát [136], az általános kumulációt.

- A legelső függvények használatakor tisztázandó, hogy a függvények bemenetén – paraméter, argumentum – az adattípusnak megfelelő adatot kell megadni, az eredménynek meghatározott adattípusa van.
- Az első átlagszámításhoz kapcsolható a hibakezelés, ebből az alternatív érték számolása, a HA() függvény.
- Adatsorokból a feltételnek megfelelő értékek külön oszlopban megjelenítése (szűrés), az erre végzett összegzés a feltételes összegzés. A feltételes összegzések függvényeinél a szűrés fizikailag nem jelenik meg, de az algoritmus elvégzi.
- Rendezés fájlkezelőkben dátumra, fájlnévre…, szövegszerkesztőben bekezdésekre. A rendezési kulcs, azaz a reláció tulajdonságai mindenképp vizsgálandók. Tananyagként táblázatkezelésben és adatbáziskezelésben tanítom. Az algoritmizálása a rendezés hatékonysága miatt fontos, mivel rendezett sorozatban a logaritmikus keresés gyorsabb, de a sorozat karbantartása nehezebb.

A különböző alkalmazásokban a feladat szövegének elemzése, a probléma részekre bontása, a feladatmegoldásnak (problémamegoldásnak) fenti a módjai mind a programozás és algoritmizálás, mind az informatikai gondolkodás fejlesztése szempontjából fontos.

A feltételes összegzések megfeleltethetők a megtanulandó algoritmusok zömének. Erre a feladattípusra az imperatív algoritmuson kívül az adatmennyiség, a kérdések száma és a feltételek összetettsége alapján különböző deklaratív megoldási lehetőségek lehetnek hatékonyak. A megoldási módok taníthatóságát, a tanítás sorrendjét és feladattípusok szerinti osztályozását elemzi a Guess the code of conditional summation [137] cikk. Legjobb több megoldási módot egyszerre bemutatni, kiemelve a megoldások algoritmusa – és ezzel hatékonysága – közötti különbségeket, a tanulóra bízva, hogy melyik módszert választja. Az algoritmikus gondolkodást leginkább a tömbképlettel történő megoldás igényli [138], a többi módszerhez a tanítás során célszerű kapcsolni a „géptől elvárt” algoritmus megfogalmazását.

Problémamegoldás informatikai eszközökkel témakörön belül, a táblázatkezelés és adatbáziskezelés tárgyalásában be kell mutatni, hogy egy probléma többféle eszközzel is megoldható, de ennél is fontosabb annak sokféle szemléltetése, hogy a megfelelő eszköz kiválasztása nem csak a probléma típusától, hanem annak méretétől – adatmennyiségétől, a megoldási mód számításigényétől –, az adathoz és a műveletvégzéshez való hozzáférés módjától, az eredmény aktualitásának fontosságától is függ. Az informatikai tudás része a hatékonyan használható eszköz kiválasztása, az eszköz hatékony (gyors, pontos) alkalmazása.

Az algoritmusok szövegalapú programozási nyelven történő megvalósítása sok esetben a deklaratív megoldások mögé képzelt algoritmusok imperatív nyelven történő kipróbálását jelenti. Az oktatás során ki kell emelni, hogy ugyanazt a problémát azért oldjuk meg többféleképpen, mert alternatív megoldások ismerete szükséges ahhoz, hogy mérlegelés után a legmegfelelőbbet ki tudjuk választani. Hasonló mérlegelés szükséges alternatív alkalmazások vagy operációsrendszerek közötti választásra is, illetve – az informatika humán területre való átvetítésével – társakkal való hatékony együttműködéshez is.

6.2.4.3 Logika

A vezérlési utasítások végrehajtása logikai kifejezések kiértékelésének függvényében történik. 1987-ben a feltételek logikai kifejezéssé formalizálását vizsgáltam [139]. Azóta is probléma, hogy a köznyelven megfogalmazott feltételek esetenként csak a közlő szándékának ismeretében, intuitíven értelmezhetők, amire egy gép nem képes [140]. A pontos megfogalmazás gyakorlása már óvodában elkezdhető. Formalizált leírás a blokknyelvekkel együtt majd matematikából és informatikából is feladat- és problémamegoldások során tanítható.

6.2.4.4 Programkészítés

A programozástudás egy lehetséges értelmezése, hogy „képes programot készíteni”. Részletezve:

1. tudjunk utasításokat kiadni, amit a gép tárol, értelmez – programozásinyelv-ismeret;
2. az utasításokat át tudjuk adni a gépnek – fordítás (interpreter, compiler);
3. el tudjuk indítani a végrehajtást – futtatás;
4. ellenőrizzük, hogy a végrehajtás a szándékunknak megfelel-e, tudjuk módosítani az utasításokat – tesztelés, szemantikus ellenőrzés
5. hiba esetén értsük a hibajelzéseket (legalább a legalapvetőbbeket) – fejlesztőkörnyezet ismerete, szintaktikai hiba és eszközhiba felismerése;
6. ismerjük a hiba elhárításának módját – eszközismeret, eszközkezelés.

A legegyszerűbben programozható (oktató) eszközök a padlórobotok. Az első algoritmus, amit meg kell tanulni: bekapcsolás, szerkesztőmódra váltás. kódolás, kód lezárása, futtatás, szerkesztőmód… kikapcsolás. A kódolás itt a megfelelő gombok lenyomása. A programozást nehezíti, hogy a „kódolást” nem szabad elrontani, de ez készségfejlesztő hatású is. Az „irányító panel” – a kódoláshoz használható kártyák, kockák és egyéb eszközök – használatával fizikailag kettéválik a fejlesztő és a futtató eszköz. Lényegében ugyanígy „készítünk” programot amikor kódban írunk weblapot vagy táblázatkezelőben szerkesztünk egy képletet.

A robotok programozása során hamar megjelenik a ciklus, a táblázatkezelőben az elágazás, a weblap kódjában az egymásba ágyazás, a prezentáció animációjában az egymásutániság és az időzített, esetleg párhuzamos végrehajtás. A lehetőségek bővülésével bővül a nyelvi készlet, a hibalehetőségek száma.

A középiskolában tanított (egyesek szerint „megtanulhatatlan") szövegalapú programozás minden ponton váltást jelent a blokknyelvű programozáshoz képest: a mutató eszköz használata helyett gépelés, egy elem beillesztése helyett helyesírás és a szintaktika ismerete kell; nincsenek előregyártott adatok; a szemantikai hibák kiszűréséhez ismerni kell a fejlesztőkörnyezet debugoló lehetőségeit; a futtatásnak többféle módját kell használni. Hasonló módon megadható a többi alkalmazástípushoz (táblázatkezelő, prezentációkészítő, weblap-készítő) képest is, hogy miben más a szövegalapú programozás fejlesztő környezete, ehhez kapcsolódva a programozási nyelv és a programkészítés.

Az informatikaoktatás során a számítógépes problémamegoldást kétoldalról közelítjük.

- A számítógépes alkalmazásokkal előállított termékek a számítógép funkcionalitását egyre jobban kihasználják: kezdetben statikusak majd animáltak végül némi intelligenciával is rendelkeznek.
- A megoldások minőségében az előregyártott intelligens komponensek összeillesztésétől haladunk a komponensek egyedi elkészítése irányába.

A két szál találkozása a vezérlési struktúrákból építkező szöveges programírás, a tesztelés és hibajavítás képessége egy, a fordítást és futtatást végző, az előbbieket intelligensen támogató fejlesztési környezetben. A programkészítésnek minden eleme valamilyen korábbi ismeret kreatív alkalmazása (└5c).

6.2.4.5 Programozás

A programozás több, mint programkészítés, mert része az adat- vagy objektummodellezés, valamint az algoritmus tervezése is és a dokumentálás, publikálás is. A programkészítés a programozás fizikai megvalósítása, miközben a programozás jelentősebb részben szellemi tevékenység.

A közoktatásban elvárt programozástudás meghatározásához a programozást a táncművészethez hasonlítom: a programozó tudásához képest az elvárt tudás olyan, mint a táncosok számára a járás képessége. A tanulónak képesnek kell lennie önállóan egy adott feladat vagy probléma esetén meghatározni a kapott adatok típusát, a megoldáshoz szükséges adat- vagy objektum-struktúrát. Meg kell tudni fogalmaznia a vezérlési szerkezetek szintjén a megoldás algorit-

musát és a kimenetnek, az eredménynek formáját. Ezek alapján el kell tudnia készíteni a programot (azt a programot, ami a feladatot megoldja), tudnia kell a megoldását tesztelni, a hibákat önállóan kijavítani. Tudnia kell a programjához felhasználói dokumentációt készíteni, amelyben leírja a program működésének feltételeit és korlátait, a kezelés módját, továbbá tudnia kell publikálni a kész programot, illetve kezelni a kódoláshoz, a fejlesztéshez szükséges forrásokat.

A programozástudás „tipegő" szintjének jellemzői[20]:

- a feladat megoldásához néhány elemi adat vagy string kezelése szükséges,
- az algoritmus bonyolultsága az egyszerű programozási tételeknek megfelelő
  - for(){if(){}} vagy
  - több ciklus egymás után vagy
  - ciklusmentesen több feltétel kombinálása;
- a feladat pontosítását követően a megoldás önálló megtervezése,
- kódolás;
- a kódolási hibák önálló javítása, futtatással tesztelése,
- az eredmény értelmes megjelenítése,
- az elkészült program bemutatása, a kész program publikálása felhasználási tájékoztatással;
- beszámoló az adatmodell, az algoritmus és kód bemutatásával a megoldás során felmerülő problémákról és döntésekről.

A programozás „tipegő" szintű teljesítése azt jelenti, hogy a tanuló programozással teljesen önállóan megoldott egy feladatot, ezzel tapasztalatot szerez arra vonatkozóan, hogy képes programozni, programozási problémát megoldani.

6.3 Az informatikaoktatás tartalmi kiegészítéseinek összefoglalása

Soloway 1993-ban jelezte [4], hogy mindenkinek tudnia kell programozni, de a tanulás és tanítás kivitelezése kérdéses. A tankönyvekhez, tanmenetekhez képest nehéz meghatározni, hogy pontosan miben más az, amit informatikából tanítok. Legutóbb vezetőtanárként azt mondtam, hogy minden órának a tananyagához meg kell tudni nevezni, hogy mi támogatja majd a programozás oktatását, mi benne a programozás. Ez elég konkrét feladatnak bizonyult a szemléletmód alkalmazásához, de nem teljesen pontos.

[20] T4 tézishez az elvárt ismeret pontosítása.

Valójában nem a programozást kell előkészíteni, hanem informatikát (4.7), az informatika tudomány aspektusait, informatikai gondolkodást kell tanítani, aminek szerves része a programozás (4.8) [123]. Hasonló szemléletű a mindennapok informatikáját népszerűsítő Bebras (e-hód) [141] verseny is, aminek gondolati alapja, hogy az informatika mindenhol jelen van.

Az informatikaoktatás során a tanított témakörök egy része – a 2020-as NAT-ban ez a 3. témakör – **célzottan az informatika oktatását tűzi ki, a többi témakör a más tantárgyakhoz szakmailag nem kapcsolódó, az információs társadalomban általánosan szükséges eszköz- és alkalmazásismeret biztosítását határozza meg. Ez a fajta kettéválasztás az informatika oktatását elnyomja, időhiány miatt sokszor negligálja.**

Az informatika oktatásának a tantárgy összes témájára való kiterjesztése az oktatási cél átértelmezését jelenti, amelyben az informatika tudományspecifikus szemléletmódját érvényesítem minden témakörre.

A már hivatkozott IX. mellékletben kivonatosan beemeltem a NAT-ban megfogalmazott fejlesztési területeket és eredménycélokat, ezeket egészítettem ki azzal, aminek a kutatásaim alapján azokban – a leírtakon felül – szerepelnie kellene. Természetesen, lehet úgy tekinteni a kiegészítésekre, hogy ez az a tudás vagy készség, amit a minimum felett lehet tanítani, de ez a szemlélet nagyon veszélyes. Nagyon röviden, egy hasonlattal élve: az egyetemi tanulmányoknál jól látszik, hogy ha a képzés célja és a tanuláshoz igénybe vett módszerek az elégséges szintet célozzák meg, akkor a bukásnak nagyon nagy a valószínűsége. A NAT-ban megfogalmazott fejlesztési területek és eredménycélok valóban elégségesek az adott oktatási stádiumban, de nagy a veszélye annak, hogy elszigetelt tudáselemek maradnak, ebből következően a feledés ködébe vesznek. Emiatt minden leírt téma hasznossága a hozzá írt kiegészítés nélkül vitatható.

A LAU-Modellt alkalmazva látható, hogy a NAT-ban megfogalmazott eredménycélok jellemzően LAU^L3, illetve későbbi hasznosítás esetén LAU^L5a-nak felelnek meg. Azaz a tanult ismeretek, fogalmak felhasználása pont úgy történik, ahogy a diák tanulta.

A tananyagot az informatikatudomány alapjainak tanítása szempontjából vizsgáltam. Azt tapasztaltam, hogy a NAT konkrét alkalmazások, eszközök adott célokra történő felhasználásának gyakorlatát írja elő. Az informatikaoktatás tartalmi leírásához ezt ki kellett egészítenem a tananyagban szereplő fogalmak és eljárások értelmezésével, modellezésével, elemzésével – azaz informatikai gondolkodás alkalmazásával. Ezek, illetve a leírtakhoz hasonló gondolatmenetű kiegészítések az új ismereteket a már tanult, tapasztalt ismeretekbe többszörösen beágyazzák, ezzel a tudást transzferábilissé teszik. Ez azt jelenti, hogy LAU^L5b szintű lesz a tudás. A

tanultak más témáknál tanult fogalmakkal, ismeretekkel való összekapcsolása, felhasználása, a LAU^{L}5c szintnek felelnek meg.

Korábbi évek tapasztalata és a kutatásom során kimondottan erre célzott megfigyelésem, hogy a LAU^{L}5a szint nem alkalmas informatikai gondolkodás fejlesztésére, általában modellalkotásra sem; nem tartalmaz értelmezést és elemzést, csak megértést, befogadást. Emiatt nem jelent informatikai tudást.

További problémát jelent, hogy ha a LAU^{L}3, LAU^{L}5a szint elérése a cél, akkor ahhoz más oktatási módszerek szükségesek, hiszen a cél eléréséhez nem kell kreativitás. Ilyenkor a tananyag átadása a hangsúlyos, a domináns oktatói gondolkodás alapján: először elmondom, majd gyakoroltatom és amikor már megy, akkor adok felmérőfeladatokat, ahol ki tudja próbálni a tudását; akinek elég jól megy, azoknak adok érdekesebb, gondolkodtató feladatot. Az eredmény az, hogy csak azok gondolkodnak, akiknek jól megy a betanított rész.

Ha a cél a LAU^{L}5b, akkor lehet, hogy nem is kell mindent elmondani, lehet a felfedezésre is hagyni valamit; a gondolkodtató feladatok megelőzhetik a felmérőt – például kutatási vagy projekt feladatként. Ezt úgy lehet például elérni, hogy sokkal korábban elkezdjük „emlegetni”, megnevezni az adott ismeretet, amikor valamilyen kontextusban feltűnik. Így mire tananyag lesz, már ismerős – LAU^{L}1A, aktívan hasznosuló – dologról lesz szó, amihez a részleteket önállóan is fel lehet deríteni.

A LAU-Modell az egyes tanulási egységek egymásra építettségét is figyeli. A NAT-ok ebből a szempontból igyekeznek útmutatást adni, de csak 2 vagy 4 éves bontásban. A tanterv 1-2 éves bontást tartalmaz. A tanmenetben elvileg megtervezzük a témák egymásutániságát, ezzel az egymásra épülést is, de gyakori, hogy a tanév során ezt valami miatt nem tartjuk be. A tananyagok egymásután szervezése nagyon sokféleképpen lehetséges, mert az ismeretek hálót alkotnak, amelyből többféle elrendezéssel lehet hierarchiát képezni. Azonban a tervezésnél figyelembe kell venni a tananyagfelépítés általános törvényszerűségeit (LAU-modellel megfogalmazva a II/2. mellékletben található), illetve a tanulást befolyásoló egyéb tényezőket.

7 Informatikaoktatás módszerei és eszközei[21]

Kutatásom az informatika és a programozás oktatásának módszertani kérdéseivel foglalkozik. Eddig két kérdést bontottam ki:

- Mi az informatika, illetve a programozás?
- Mit tanítsunk informatikából, illetve programozásból a közoktatásban?

A következő kérdéskör a fentebb meghatározott tartalmak (és kompetenciák) oktatásához alkalmazott eszközök és oktatási formák. A központi szabályozók és szakmai elvárások, célok figyelembevételével, amennyiben a diákok tudása engedi, a helyi tantervben és az óravázlatokban lehet bővíteni a tananyagot, de a minimumot teljesíteni kell. Az oktatási gyakorlatban is az a jellemző, hogy az alapok megtanításán van a hangsúly, így minden arrafelé visz, hogy az oktatás módszereinek megválasztásakor is az alapok elsajátítása legyen a középpontban, ne az erre épülő tehetséggondozás. A kutatásom részeként részletesen elemeztem több egyetemi tantárgy, közülük legalaposabban a *Programozási alapismeretek* (ELTE IK) és a *Programozás alapjai* (BME VIK) tárgyak tananyagfelépítését, mivel ezek a programozástanulás mindenki számára előírt, alapozó, kezdeti fázisát jelentik az egyetemeken, ahol „nulláról kezdik a programozás tanítását”.

7.1 BME VIK Programozás alapjai 1 LAU-modellje, tanmenet, óratervezési kérdések

A tananyag feldolgozására 14 hét, hetente átlagosan 15 óra tanulást számolnak a tárgyleírás alapján, ebből 6 óra kontaktóra, 4 óra rendszeres tanórai felkészülés, 5 óra projektszerű felkészülés (ZH-ra, beadandóra). Összesen 210 óra.

Mint azt [77] cikkünk prezentációjában részletesen bemutattuk, a tervezés során az egyes tudáselemek LAU^L1–3 szintű elsajátítása történik az előadásokon, gyakorlaton és laboron – pontosabban előadásokon, laborokon majd gyakorlatokon, – a további, magasabb készségeken történő elsajátítás házi feladat. Ezzel szemben a közoktatásban úgy kell számolni, hogy a rendszeres tanórára felkészülés helyett tanórai gyakorlást kell szervezni, a projektszerű felkészülés lehet csak házi feladat. Természetesen ez nem jelenti azt, hogy a ZH, illetve dolgozat előtti héten kellene csak otthon tanulni, azt viszont jelenti, hogy a közoktatásban a gyakorlás men-

[21] A T1 és T2 tézisekben megfogalmazott sikeres, minőségi és hatékony oktatáshoz módszertani elemzés, módszerek kiválasztása

torált, tanár által segített, míg az egyetemen nem; a hallgató magára van hagyva. Ebből következően, a 210 óra közoktatásban 140 tanóra és 70 óra házi feladat megosztásban lenne lehetséges. Ez iskolai rendszerű oktatásban egy év alatt heti 4 órát jelentene, de heti 1 órával számolva – ami a leghaladóbb szemléletben óraszámként szerepel – 4 teljes tanév tananyagát jelenti. A LAU-modell jellegzetessége, hogy az időt is figyelembe veszi. Míg a közoktatásban informatikából a tanulás elhúzódása, közben a felejtés jelent gondot, az egyetemen a feldolgozás nagy tempója az elsődleges probléma. Az egyes fogalmak nem tudnak „megérni", rögzülni, máris másra kell figyelni, mást kell tanulni. Azok a tudáselemek, amelyekre sokáig nincs szükség, szelektálódnak. De azokra a tudáselemekre sem lehet számítani, amelyek még nem rögzültek. A *Programozás alapjai 1* tárgy tananyagfelépítése nagyon átgondolt, de néhány probléma minden tervezettség ellenére felmerül. Ezek a problémák az informatikaoktatás tervezésére is jellemzők, ezért módszertani vizsgálatuk indokolt.

7.1.1 Sok kicsi LAU sokra megy

A megállapítások hátterét, esetek bemutatását a X/1. melléklet tartalmazza.

7.1.1.1 Előrehivatkozás

Az alapoktól felfelé strukturáltan építkezésnél néha elkerülhetetlen, hogy megértés nélküli használatot várjunk el. Például: scanf(″%d″, &a) & jelölése. Tervezés szempontjából: a LAU^{L}5 megelőzi a LAU^{L}1-3-at. A tanítás során az ilyen típusú kényszereket nagyon alaposan kell tisztázni. Hangsúlyosan ki kell emelni, hogy az adott dolgot „most még nem kell érteni; ne akard érteni, csak használd; <ekkor> fogjuk tanulni". Természetesen minden ilyen kereszthivatkozás gyengíti a logikus tanulást, ezért a tervezés során törekedni kell a „majd később" elemek minimalizálására. A projektszerű, felfedeztető tanítás esetén a tanulási útvonalat az ehhez hasonló ismeretlen elemek adják, ahol épp az a feladat, hogy „most állj meg és értsd meg."

7.1.1.2 Ritkán előkerülő, de fontos tananyagelemek

Lehet úgy tervezni, hogy első előforduláskor megtanítunk egy tananyagelemet, majd erre hivatkozunk, de gyakorlatilag addigra elfelejtődik, taníthatjuk újra. Számítani kell rá, minden esetben újra meg kell tanítani, minden esetben más, újabb kapcsolódást véve a fogalomhoz. Ez azt jelenti, hogy LAU^{L}1 után újra LAU^{L}1 szinten ismerkedik a fogalommal a tanuló, egy-egy vonatkozása tekintetében ki is próbálja (LAU^{L}1–3), de amíg ezek az ismerkedések nem érik el a LAU^{L}5a szintet, addig nincs mit számonkérni (még beugróban sem). Ezzel a módszerrel lehet előkészíteni nehéz, sok kapcsolattal bíró fogalmakat is. Lényegében így tanulunk beszélni is és így lehet sokéven át, de hetente nem sok órában építkezve tanítani az informatika fogalmait is. A közoktatásban azonban problémát jelenthet, hogy a sok kis részletet nem szabad osztályozni,

mert az épp úgy akadályozza a tanulást, ahogy egy regényből a fejezetenként írt olvasási napló a regény élvezetét.

7.1.1.3 Alternatív megoldások

Az alternatív megoldások újabb tervezési feladatot adnak. Cikkek [pl. 144] szólnak arról, hogy a vezérlési struktúrák közül melyiket jobb előbb tanítani (az elágazást is hozzávéve).

Általánosan elmondhatjuk, hogy – főleg informatikából – egy problémára, feladatra gyakran többféle megoldás létezik, de hasonló problémák esetén a lehetséges megoldásokból csak néhány használható. Máskor a megoldási módok (szinte) ekvivalensek. Ilyen tananyagtartalmak esetén érdemes az összes lehetséges megoldást egyszerre vagy időben egymáshoz nagyon közel megmutatni, majd ezt követően akkor, amikor mindkét (vagy több) megoldási lehetőség helyes, a megoldásokat váltakozva alkalmazni. Természetesen, az egyszerre bemutatás csak arra az esetre igaz, amikor mindkét megoldás helyes, szokásos. Azokban az esetekben, amikor egyértelmű, hogy az alternatív megoldások közül melyik a jobb, akkor csak azt említjük vagy, ha már ismert a másik megoldás is, akkor indokolni kell annak az elvetését. Így érhetjük el azt, hogy azt is megtanítsuk, hogy az alternatív megoldások közül milyen szempontok alapján választunk; ezzel tanítunk a programozáson belül is informatikát, a programkészítéshez kapcsolódóan gondolkodásmódot.

Az alternatív megoldások egyszerre tanításával mindegyik megoldási lehetőséget hatékonyabban tudjuk tanítani, mert a formális befogadáshoz kapcsolódik az értelmezés, az elemzés, az informatikai gondolkodás. A közoktatásban talán a legtipikusabb a már korábban említett feltételes összegzések több megközelítési módjának ismertetése [137]. A problémamegoldás eszközeinél alapvető fontosságú az alternatív megoldások megmutatása, de a digitális írástudás során is a rasztergrafikus vs. vektorgrafikus ábrázolás; a szövegszerkesztés WYSiWYG vs. kódnézet alternatívái; a hardvereszközök kezeléséhez kapcsolódóan az egérrel vs. billentyűkombinációval végzett művelet választási lehetősége és a kommunikáció formái közötti alternatívák tanítása rejti a lényeget, az informatikai tudást.

7.1.1.4 Rutinok

Az alternatívák tanításának van egy elég kellemetlen oktatói vetülete: a gyakorlatban kialakult szokásokat tanítás közben kontrollálni kell vagy inkább kellene. A tanítás során bemutatott alternatív megoldások közül a tanulók saját megoldásaikhoz kiválasztanak egyet vagy néhányat. Cél az, hogy idővel minden megoldási lehetőséget kipróbáljanak és a tapasztalat alapján kialakítsák saját szokásaikat. Eközben az oktató folyamatosan a sokféleséget gyakorolja, azaz a rutin helyett az motiválja, hogy ne maradjon ki az a megoldás se, amit ő nem szokott választani. A

legtipikusabb problémák az operációsrendszer, az alkalmazói szoftverek, a programozási környezetek és nyelvek, a billentyűzetkiosztás választása során jelentkeznek.

Az informatikai tudásunk haszna, hogy rutinokat alkalmazunk, ezt oktatóként sem adjuk fel. Emellett az alternatív megoldásokat be kell mutatni és azokat a megoldási lehetőségeket is értékelni, elemezni kell, amelyeket nem használunk rutinszerűen. Sok esetben a használt megoldásunk nem a legoptimálisabb, de mivel rutinosan használjuk, hatékonyabb, gyorsabb lehet számunkra, más megoldásokhoz képest. A rutintól való eltérés megsokszorozza a hibalehetőségeket. Ezt láttuk a [59] felmérésünkben is (nem a megszokott változónév használata), de ugyanitt az is kimutatható, hogy a rutin néha zsákutca (tesztesetek gyártása, ami nem volt feladat). Nem csak gazdasági [26, 28], hanem informatikai feladat megoldásában rendszeresen döntenünk kell a rutin (default) és az egyedi megoldás között.

Az alternatív megoldások tanítása az adott tananyag LAU^L5b szintű tudásához szükséges. Ebből következik, hogy egyetlen megoldás kiválasztása és állandó használata legfeljebb LAU^L5a szintre elegendő. Egy rutin megtanulása után – másik évben, másik tantárgyban, esetleg csak az élethosszig tartó tanulás során – lényegében újratanulással lehet az alternatív megoldást is megtanulni, ekkor hozzákapcsolni a már létező rutinmegoldáshoz. Ezt követően akár módosulhat is a rutin. Ilyen esetekben a tanulást úgy tervezzük, hogy az első megoldást LAU^L1-5a szinten tanítjuk, a második és többi megoldást LAU^L1-5b szinten, az előzőkkel összevetésben. Az alternatív megoldások közötti kreatív választás jelenti a LAU^L5c szintet.

7.1.2 Tanulás szervezése

A BME VIK tanítási specialitása a gyakorlat és labor kettéválasztása. Ez és az órarendszervezési anomália tette lehetővé, hogy összehasonlítsuk a korábban megszokott előadás, gyakorlat, labor sorrendet a hatékonyabb, előadást követően labor, majd ezután a gyakorlat megoldással. Az új sorrenddel az előadás LAU^L1 funkcióját a laboron a kipróbálás LAU^L2 követi, ezután a gyakorlaton a LAU^L3 szinten a tanultak belső összefüggéseinek vizsgálata következik. Azonban ezzel a szervezett tanulásnak vége is volt. A gyakorlás és ezzel a LAU^L5 megfelelő szintjének eléréséhez a hallgatók nem kaptak ösztönzést, csak ömlesztett feladatokat. Az önálló továbbfejlesztés motiválására készült a *Haladási napló*. Ebben a hétről-hétre megjelenő fogalmakhoz a tudás minőségét jellemző, 0-tól 5-ig szintezett állításokat írtam. Ezek közül jellemzően a 3. jelentette azt a szintet (LAU^L5a), amit a gyakorlat végére el kellett érni, a 4. szint elegendő volt a ZH-ra (LAU^L5b), az 5. szint pedig a fogalom egy magasabb szintű értelmezése vagy új felhasználási köre volt (LAU^L5c).

7.1.2.1 Haladási napló

A napló beválásáról a 2017-es ISSEP konferencián és az InfoDidact konferencián számoltunk be [146, 147]. A hallgatók többsége rendszeresen ellenőrizte haladását, következő évben az oktatók számára is olvashatóvá tettük az értékelést és az értékek mellett jelöltük a legutóbbi módosítás irányát is. A *Haladási napló* ezzel a kiegészítéssel segítette a hallgatót az elvárások megismerésében, a saját tudásának értékelésében, miközben az oktatója – e-mail és dolgozat nélkül – láthatta, ha a hallgatónak vélhetően problémája van, segítségre szorul. A hiányzásoktól, jelenléttől függetlenül látható volt a tananyagban haladás iránti érdeklődés elveszítése. Ennek oka lehetett az is, hogy a hallgató messze előrébb járt, ezért nem jelentett kihívást a haladás, azonban ez könnyen ellenőrizhető. Másik ok a lemaradás, a tananyag követésének feladása, ami a kimaradás előtt néhány alkalommal már a beírt, majd nem beírt vagy hirtelen romló értékelésekből látszik. A régebben beírt értékelések lerontása szintén figyelmeztető jel, ahogy a ZH előtti időszakban a sok 0–3-as önértékelés a számonkért témákból.

7.1.2.2 Ellenőrzés, gondolkodás, érvelés

Az egyetemi hallgatókkal szemben elvárás, hogy házi feladatként önállóan próbálják a feladatokat megoldani. Laboron kérhetnek segítséget a megoldásokban, ha úgy érzik, hogy problémájuk adódott. A gyakorlatokon – némi önálló munka után – jellemzően a gyakorlatvezető mutat meg egy helyes megoldást. Csak az agilis hallgatók gondolkodtak el tényleg a megoldásokon. Jellemző, hogy kivárják a hallgatók a helyes megoldás bemutatását. Az önálló munka, a megoldás indoklása, a szakkifejezések szóbeli használata, az eltérő megoldások összehasonlítása lényegében hiányzott a gyakorlatokról. Ezért vezettük be az *Ellenőrző feladat*ot. Ennek a feladatnak az eredménye sem számított bele az értékelésbe, de a feladat önálló megoldását követően, mintha ZH feladat lett volna, társértékelést kértünk. Az értékelés, pontozás megbeszélését követően az eredményt és a megoldással kapcsolatos észrevételt a portálon be lehet jegyezni.

Bár – a visszajelzések alapján – a társértékelés sokszor csak önértékelés, de önmagában a pontozás, az ezzel kapcsolatban felmerülő és megbeszélt kérdések, problémák hasznos útmutatást jelentenek a hallgatóknak. Számos „hoppá, jobban oda kell figyelnem a részletekre”, „látszik, hogy nem tudtam eléggé felkészülni”, „azt hittem ez megy” jellegű bejegyzés bizonyítja, hogy a tárgy teljesítésébe beszámító mérés előtt érdemi visszajelzést kapott a hallgató.

A közoktatásban az itemekre pontozott vizsgák – jellemzően az érettségi vizsga – esetében ugyanezt a módszert alkalmazom a felkészítés során. A kiadott feladatsor önálló megoldása

után a tanórán legalább kezdetben társértékeléssel pontozzák a diákok a munkájukat, eközben az alternatív megoldások elfogadhatóságát és a problémákat is megbeszéljük.

Az ilyen típusú – közös, megbeszélős – ellenőrzésnek a gondolkodás szempontjából nagyon fontos szerepe van. A programozás és az informatikaoktatás során jellemző, hogy nem a tanuló tevékenységét, gondolatait, tudását látjuk, hanem – indirekt módon – annak eredményét. Az eredményből próbálunk visszakövetkeztetni arra, amiből az született, hogy útmutatást tudjunk adni a fejlesztésre, további tanulási feladatra. Az értékelési szempontok és a mintamegoldás az indirekció miatt sokszor nem elegendők ahhoz, hogy a tanuló megértse, milyen minőségű a megoldása. „Ezer oka lehet annak", hogy a tanuló megoldása eltér a kiadott megoldástól. Lehet, hogy egyetlen karakternyi eltérés nagy hiba, míg egy teljesen másképp megírt megoldás tökéletes. Tipikus értékelési problémák:

- Nem minden működő megoldás helyes. A BME-n a memória fel nem szabadítása, a túlindexelés (lezáró nulla elfelejtése) a tipikus probléma, az ELTE-n a nem strukturált eszközök használata.
- Attól, hogy működik a program, még nem biztos, hogy minden esetben helyes a működése. Erre a szóközökkel középre igazított címtől kezdve, a kivételkezelési problémákon át, a változók konstansként bedrótozásáig rengeteg példát láthatunk.
- Egy feladatnak többféle megoldása is lehet. Attól, hogy a megoldás a mintától eltér vagy az értékelési szempontok az adott megoldásra nem értelmezhetőek, még lehet jó a megoldás.
- Egy feladatnak számos olyan megoldása lehetséges, ami jó, de nem „illendő" az adott tudás-tartományban ilyen megoldást adni. Például a megoldásban felesleges műveletek vagy kiírások vannak, a struktúra homogén kollekciója helyett az adattagok egymástól független tömbjeiben tárol adatot.
- A feladat többféleképpen értelmezhető, a megoldás nem arra az értelmezésre készült, amiről az értékelés szól.

Az értékelési problémák felvetése minden esetben a tanulást segítik, alkalmat adnak arra, hogy a tanuló kimondja (hangosan, artikulálja) gondolatait. Így lehet kiszűrni a félreértéseket, megkülönböztetni az elírást a téveszméktől, illetve így lehet észrevenni, ha a hallgató logikus gondolkodás – szabályok alkalmazása – helyett „mintaillesztéssel" oldja meg a feladatokat, ami egyik jele lehet a magolásnak.

A feladatok megoldása jelezhet LAU^{L}3–5a tudást, azonban az érvelés, a megoldási módokról gondolkodás, kommunikálás a feladattal kapcsolatban a LAU^{L}5b–5c tudásszintjéhez tartozik.

A programozáshoz – és általánosan is egy informatikafeladat megoldásához – többféle memorizálás szükséges [25]. Feladatmegoldást lehet úgy gyakorolni, hogy sok adott típusú feladatot megoldunk. Amíg nem megy, megnézzük a megoldást és azt reprodukáljuk. Az ilyen típusú gyakorlás lényegében memorizálás, tesztfeladat megoldása [28]. A dolgozat sikere azon múlik, hogy felismerjük-e a típust és megvan-e a rutinunk a megoldás reprodukálására. Ez a tanulási stratégia sokáig sikeres, azonban a bemagolt rutinok kopnak, elfelejtődnek miközben számosságuk növekszik. A tudás ilyen módon LAU^{L}5b szintig emelkedhet, ami a betervezett vizsgákra elegendő, de a felejtés miatt később nem használható.

A feladatok megoldásának indoklása (védése) során nem szabad elfogadni az „ezt vettük előző órán" jellegű mintaillesztést. Nem érv a minta forrásának megadása, a megoldáshoz szükséges döntéseket kell elmondani. Mintha bizonyítás lenne, fel kell építeni a feladatot. Ez az, ami képessé teszi a tanulót az aktivizálható tudás megszerzésére, mivel az idővel elfelejtődött részleteket képes lesz reprodukálni. Sok ilyen típusú gyakorlással a tanuló a gondolkodás reprodukciójából szerez rutint. Ez a fajta feladatmegoldás a LAU^{L}5c szintet jelenti a megoldáshoz felhasznált tudáselemekre, miközben az épp tanult témában a LAU^{L}5b szint hosszabb távon is elérhető marad. Hétköznapian: nem csak a két héten belüli dolgozatban, hanem akár a következő évben is képes lesz a tanuló az adott feladattípus megoldására.

A tanulási módszerek közötti különbség a várható eredményesség miatt a *Programozás alapjai 1* tantárgy honlapján olvasható ZH tanácsok között is előkelő helyen szerepel[22]

> *Hogyan kell tanulni a prog ZH-kra? „Sehogyan." Ez nem bemagolható, csak gyakorolható tárgy.*
>
> *…A programozás inkább a KRESZ-re hasonlít. … ha a KRESZ-hez megtanulod ezt a két dolgot: aki szemből jön, azt elengeded, és aki jobbról jön, azt is elengeded – akkor szinte mindent tudsz. A képes kérdések 98%-ára ebből a két szabályból ki*

[22] https://infoc.eet.bme.hu/nzhtanacs/ [2021.10.30]

tudod következtetni a választ. Ha még egy kis gyakorlatod is van, akkor kérdésenként semmivel nem több, mint két másodperc alatt.

A tárgy gyakorlatain kiadott „*Ellenőrző feladat*” a lemaradások jelzésén túl ezt a fajta tanulási módszert hivatott ellenőrizni. Ezért javasoltam az oktatóknak, hogy a többi feladatot is hasonló módszerrel próbálják feldolgozni, az oktatói magyarázatoknak inkább csak a viták mederben tartása, központi elvárások közvetítése legyen a célja.

7.2 ELTE Programozás (alapismeretek) 1 LAU-modellje

Az ELTE-n korábban *Programozási alapismeretek*, 2018 óta *Programozás* néven futó tárgy LAU-modellje teljesen más, mint a BME-s megfelelő tárgyé. Az ottani határozottan lineáris felépítéshez képest az ELTE-n ebben az egy tantárgyban párhuzamosan három nyelven, három kifejezésformában kell leírni a megoldást, az ismeretek bővítése három, egymáshoz is többszörösen kapcsolódó szálon fut. Az alapvetően programtervezést és algoritmizálást tanító tárgy legnagyobb problémája, hogy amit nem ismersz, azt nem lehet módszeresen tervezni. A programtervezéshez szükséges lenne a programkészítés ismerete, de ennek meglétére nem lehet számítani. A tervezés a deklaratív specifikációval kezdődik, ehhez egy matematikai (kicsit logikai-halmazelméleti, kicsit funkcionális) leíró nyelv tartozik. A specifikáció alapján készül a program modellje, struktogramja, amely a megoldási módszert, algoritmust írja le szabványos (rajzos-szöveges) formában. Ezt követi a gyártás, azaz a kódolás, programkészítés.

Megpróbáltam a *Haladási naplót* alkalmazni a tantárgyra, de nem volt értelme. A BME-s témákból 4-5 lecserélésével lefedhető a tananyag, de nem hetente, hanem összesen két részletben kellett volna megjeleníteni. Így egyszerre sok kérdés került ki, amiből a tényleg kezdők – akiknek a haladását nagyon kellett volna figyelni – tipikusan 1-2 értékeléseket adtak. Ez eléggé elkeserítő ahhoz, hogy ne motiváljon. Felesleges is volt, mert alternatívaként az online értékelésű, kötelezően megoldandó feladatok is mutatták az éppen aktuális problémákat.

A tananyag időben szintén három részre bontható. Az első részben az 6. fejezet 6.2.4.5 Programozás részében leírt „tipegő” szintű programot kell specifikálni, modellezni és készíteni. Ez a LAU^{L}1–2 szintet jelenti mindhárom területen. A második részben – a tömb és a struktúra adatspecifikációja mellett – a programozási tételek felismerése, modellezése és kódolása a feladat. Lényegében rutint kell szerezni a tételek felismerésében és alkalmazásában, ami LAU^{L}3–5a szintnek felel meg. A harmadik rész a tételek összetett alkalmazása, már a nevéből is látszik, hogy LAU^{L}5a–5c az elvárt tudásszint ugyanarra, amit a félév elején tanultak.

A BME-s tantárgy felépítésében egyértelmű volt az előadásnak, labornak, gyakorlatnak és házi feladatnak a tudás elmélyítésében játszott szerepe, ehhez lehetett igazítani az oktatás eszközeit. Az ELTE-n az előadás, a tantermi gyakorlat és a gépes gyakorlat nem igazodik a tudásszinthez. Vannak ilyen-olyan típusú órák, amibe a lehetőségekhez mérten be vannak osztva tananyagok. Az előadáson a specifikáció és modellezés van előtérben, de azért a programkészítésre is történnek utalások. Ahol van tantermi gyakorlat, ott a specifikáció és modellezés gyakorlása történik, a géptermi gyakorlaton pedig mindhárom témát kellene venni – azaz gyakorolni kellene a programozást, de a beugró a specifikációról és modellezésről szól. A kötelező házi feladat is jellemzően programkészítés, mivel azt lehet legjobban ellenőrizni. (Itt az ellenőrzésen az önellenőrzést és tanári ellenőrzést is értve.)

A BME *Programozás alapjai 1* tárgy folyamat-vezérelt, a tárgy oktatói, demonstrátorai a központilag kiadott tananyagot tanítják és illik betartani a központilag javasolt módszert. Az ELTE *Programozási alapismeretek* bemenet- és kimenet-vezérelt, az előadások tartalma, a határidőre teljesítendő központilag kiadott házi feladatok, a félév végén központilag értékelt évfolyam ZH-k szabályozzák az oktatást. Ezen túlmenően az oktatók, óraadók tervezik az oktatás tartalmát, alkalmaznak oktatási módszereket. A kétféle vezérlés közötti különbséget jól jellemzi az, hogy a *Haladási napló*t minden egyeztetés nélkül ki tudtam próbálni az ELTE-n, ugyanakkor az ELTE-n végzett tananyagszervezési kísérleteimnek még a csíráit sem vittem át a BME-re, mert ez ott a tantárgyfelelős feladata.

Az ELTE-n három évben, három különböző módon szerveztem a gyakorlatok tananyagát. Az első évben a programkészítés, a programozási kérdések voltak a gyakorlat témái. A specifikáció és a modellezés csak az előadáshoz tartozó házi feladat és beugró kérdés formájában került elő. Ez a módszer a specifikációt és a modellezést számonkérő ZH-knál hozta nehéz helyzetbe a hallgatókat. Láthatóan olvasással tanultak, nehéz volt reprodukálni az olvasottakat; struktogramot lerajzolni, a jelöléseket környezethelyesen alkalmazni. A második évben gyakorlatokon írtunk specifikációt és rajzoltunk modellt. Főleg én írtam a táblára, a hallgatók esetlegesen a jegyzeteikbe. Ettől kevesebb lett a programozásra fordítható idő, de jobban tisztázódtak a feltételek, valamivel jobban tudtak készülni a ZH-kra. A harmadik évben házi feladatként szótárt kértem, azaz a nyelvi elemekből és ábrák sablonos megvalósításából puskát kellett készíteniük. Emellett a beugróban nehéz specifikációs feladatot adtam, de az elfogadás szintjét alacsonyra tettem, az óra folytatása a helyes megoldás megbeszélése volt. Az órák utolsó harmadában már a hallgatóknak kellett táblára írni/rajzolni a megoldást. Ez a módszer leginkább

a BME *Ellenőrző feladat*ához hasonlítható, az eltérő értékelési rendszer miatt voltak módosítások. A módszer eredményes volt: a hallgatók aktívan, önállóan tudtak specifikációt írni, struktogramban az algoritmust megfogalmazni.

Mivel a kísérletek kis mintán történtek, a számszerű eredmények nem jelentenek túl sokat a csoportok eltérő minősége (nappali, esti) és változó összetétele miatt, valamint azért, mert közben változtak az értékelés szabályai, sőt, nappalisoknál az óraszám is. Az eredménytől függetlenül (bár ezek sem rosszak), az egyik módszert azért tartom sikeresebbnek a másiknál, mert a hallgatók saját képességeikkel jobban tisztában vannak és tudatosabban tudnak tenni a céljaikért. *Nem az a cél, hogy jól magyarázzak, hanem az, hogy jól – gondolkodva, kreatívan, problémák megoldására készen – tanuljanak a hallgatók [32, 33].*

A feladatokról való kommunikáció – akár segítségkérés, akár ötlet elmondása formájában – az informatikaórákon is hangsúlyos szerepet kell kapjon, a hatékony oktatás alapvető eszköze [35] [36].

7.3 Az egyetemek programozást bevezető tárgyainak oktatásmódszertani elemzése, leképezése a közoktatásba

Mindkét szakon részletesen elemeztem a programozás oktatásának módszereit. Itt most csak a közoktatási informatika szempontjából szignifikáns következtetéseket sorolom fel. A kiemelt hivatkozások az adott szempont szerinti elemzés mellékletére mutat.

Felmértük a hallgatók belépéskor meglévő ismereteit, a hozott tudást (X/2. melléklet). A válaszoknak az első féléves programozásban eredményekkel vett korreláció-mátrixát vizsgáltuk 2015-2017-ben a BME mérnökinformatikus hallgatói körében és 2016–2018-ban az ELTE *Programozási alapismeretek*et tanulók körében. 2017-ben a 2015-2016-os eredményeket mutattuk be a DidMatTech konferencián [148]. A kérdőívre adott válaszok és a félévi eredmény alapjául szolgáló pontszám korrelációja alapján mérnökinformatikusoknál az emelt szintű érettségi programozás feladatán elért pontszám és a középiskolai fizikajegy volt legerősebb korrelációban a tárgy pontszámával. A programtervezőknél az érettségi programozás feladata pontszámát követően az adatstruktúrák és a tanult programozási tételek vezették a korrelációs listát.

A korrelációk sorrendjében is, de a tantárgyak tematikáinak összevetése alapján [54] is látható, hogy az egyetemek az adatstruktúra és algoritmus ismeret egyike mentén szervezi a tananyagot, a másik ismeretét elvárja. Azt is láthattuk, hogy a különbség nem csak a képzési tematikában, hanem gondolkodásmódban is megnyilvánul (4.2). Sőt, a programírási szokásokat

vizsgáló kutatásból [59] az is kiderül, hogy az informatikatanárok a programtervezői szemléletű programozást tanítják.

Az egyes képzéseken gyakran találkozhatunk olyan hallgatóval, aki előzetesen a másik típusú programozást tanulta, másként gondolkodik a programozásról. A programozás mibenlétének félreértelmezése (X/3. melléklet) gyakran alapja a hallgató-oktató konfliktusnak.

- A BME-s „problémás" hallgató szemtelen, a C helyett C++ nyelven szeretne programozni, mert ott van string, és vector, nem érti miért kell a pointerekkel foglalkozni. „Persze megbukik a ZH-n, mert a csillagokat összevissza írogatja".
- Az ELTE-s „problémás" hallgató laza, nem rajzol struktogramot (azt sem tudja, hogy mi az), vagy ha mégis, az elágazásba beleírja a break-et. „Persze megbukik a ZH-n, mert nem érti meg, hogy mi a feladat, nem jön rá, melyik tétel kellene. Kavar összevissza."

A probléma lokális megoldása a régi ismeret és a jelenlegi követelmények viszonyának meghatározása. **A probléma** hosszabbtávon **a közoktatásbéli minőségi informatikaoktatással oldható meg, az informatikai gondolkodás megfelelő fejlesztésével**. Nincs jó és rossz nézőpont, csak más nézőpont. **Azt kell a diáknak majd a hallgatónak megértenie, hogy a programozás különböző helyzetekben más szabályok mentén lesz helyes.**

7.3.1 „Nulláról induló tananyag"

Az egyetemi tájékoztatók egyik kedvenc szófordulata a „nulláról induló tananyag" (X/4. melléklet). Valójában az alapoktól indul – és ez nem csekély különbség. A programozás oktatása módszerének alapkérdése, hogy mi a felépítésnek a fókusza. A programozás oktatásának módszereiről írt tanulmányokban [149, 150] különböző módszereket nevesítenek a programozás oktatásához. A tanítás során a megtanulandó ismeretek felépítése erősen utal a deduktív gondolkodási[23] mód preferálására. Az ELTE IK *Programozás alapismeretek* tantárgya algoritmus-orientált felépítésű, deduktív gondolkodási módot követ, az elemi algoritmusokból építi fel az összetettebbeket. A BME VIK *Programozás alapjai* tárgy elsősorban adat-orientált (objektum-orientált) felépítésű, szintén deduktívan, az elemi adatokból építkezik a bonyolultabb felé. A bemeneti ismereteket feltérképező kérdőív első kérdése az előzetes programozás tanulásra kérdez rá. A válaszoknak a féléves eredménnyel való összevetése (mellékletben 38. ábra) mutatja, hogy a tárgy teljesítésének esélyeit a programozási előismeretek növelik. Továbbá az is kiderül, hogy a 2009-esről a 2013-as kerettantervre áttéréssel a közoktatásban szerzett előis-

[23] Az induktív és deduktív ismeretszerzésről egy leírás: http://mmi.elte.hu/szabadbolcseszet/mmi.elte.hu/szabadbolcseszet/index3cd9.html?option=com_tanelem&id_tanelem=475&tip=0 [2021.10.30]

meret hasznossága a külsős képzésekhez képest csökken. Az ELTE programtervező informatikus nappali, esti és fejlesztő „tagozatai” külön vizsgálata több szempontból tanulságos. (mellékletben 39. ábra). A programtervezőre jobb előképzettségűek érkeznek, mint a programfejlesztőre, az esti tagozat kétpólusú: egy részük előképzett, csak papír kell; mások előismeret nélküli pályaváltók. Az adatok alapján **az „alapoktól indulás” feltétele, hogy előtte a mindennapok szintjéről lejussunk az alapokig.**

Ezt támasztja alá, hogy más egyetemek (Columbia University, Harvard College, University of Cambridge [24]) a programozás elméleti alapjainak megismerése előtt ajánlanak, vagy tanítanak (3–6 hét) egy bevezető nyelvet, ezt követi az oktatási célnak megfelelő nyelv.

7.3.2 Tanítás eszköze – gépen vagy papíron

A közoktatásban az informatika „gyakorlati tárgy”, természetesen számítógépen történik az oktatás, bár sokan füzetbe is készítenek jegyzetet. Az egyetemeken a szakma elméletét tanulja a hallgató, hagyományosan jegyzeteket készítve, amit ajánlatos papírra készíteni.

A kódolás tanítása a BME_VIK-en (X/5. melléklet) tipikusan papíralapú. A *Haladási napló* adatainak elemzésekor [146, 147] tűnt ki, hogy az összes vizsgált évfolyamon a „Debug” szempontra adott értékelések nem korrelálnak az elért eredménnyel, a gépírás negatív korrelációt mutat. A felmérésben tizenhét tudáselem önbevallásos ismeretét vetettük össze tizenkét mérés eredményével. A korreláció értékek két összetevője a hallgatók önértékelése a tudásukról, illetve a tudáselem megjelenése a mérésekben. A nagyon kicsi pozitív, illetve negatív értékek vagy azt jelentik, hogy a hallgatók nem értik, hogy mit kell tudniuk, vagy azt, hogy a mérések nem mérik az adott tudáselemet. Jelen esetben az ok: csak szóbeli elvárás ez a két tevékenység, nincs rájuk szükség a tárgy teljesítéséhez. A papírra írt kód az előadás szemléletmódjával ellentétes. A papírra jegyzetelést az támasztja alá, hogy néha rajzot is kell készíteni.

További jellemzők: a tábla-krétás előadásokon rendszeres a törlés, ami sem írásban sem rajzon nem követhető; a kivetítős előadásokon jellemzően nincs megosztva előzetesen a prezentáció és jegyzeteléshez a vetítés túl gyors; a gyakorlatokon megoldott vagy demonstráció során továbbfejlesztett feladatokban a kód változása lekövethetetlen papír alapon. Ezért a hallgatók órákon általában inaktívan figyelnek. (Néznek, mint a moziban.) Kiemelten káros, hogy a dol-

[24] Columbia: http://www.cs.columbia.edu/~bauer/cs3101-1/; [2021.10.30]
Harvard: https://online-learning.harvard.edu/course/cs50-introduction-computer-science, https://www.youtube.com/watch?v=u-kH-5JJSgU; [2021.10.30]
Cambridge: https://www.cst.cam.ac.uk/admissions/undergraduate/faqs, https://www.cl.cam.ac.uk/teaching/1920/cst.pdf [2021.10.30]

gozatokat – ahol programkódot kell írni – papíron kérik, így még gyakorolni is kell egy szükségtelen tevékenységet, hibás kódolási technikát. Hibás, mert papírra a kódot lineárisan írják, a programozásban viszont a blokkok egymásba ágyazása a helyes technika.

A specifikáció és algoritmizálás az ELTE IK-n (X/6. melléklet) szintén papírra történik. A specifikáció esetén a matematikai kifejezések, egyedi jelölések nehézkes bevitele, a modellezésnél a struktogram elkészítésének gépi megvalósítása okoz nehézséget. Itt is jelentős hatása van annak, hogy a dolgozatban hogyan tudják majd számonkérni a tudást. Mind a két absztrakcióra jellemző, hogy

- hiányzik az oktatás megfelelő digitális eszköze (a suszter cipője…),
- körülményes a gépi megvalósítás
- szemléletes, tömör, áttekinthető egy szépen kivitelezett megoldás
- a kézzel írt megoldás ritkán lesz szép és áttekinthető, inkább a javítások jellemzik
- szép megoldás mégis csak gépen készül, de nem a program tervezéséhez, hanem a kész program dokumentációjához.

Ha lenne megfelelő oktatási eszköz, amivel dokumentumon belül könnyen szerkeszthető egy matematikai kifejezés és könnyen megrajzolható a struktogram, akkor nem csak papírmentesen, de a célnak megfelelőbben lehetne oktatni.

A jelen és a jövő az informatikus számára papírmentes (X/7. melléklet). Az informatika szakmában, a távmunkában, a mesterséges intelligencia alkalmazásokban, a dolgozatok értékelésében a papírra vetett gondolatok elvesznek. A tanulásban nemhogy számítógépes, hanem tovább lépve, mesterséges intelligencia botok értékelik majd a hallgatók munkáit [145].

7.3.3 Egyetemi képzési formák, oktatási elvárások

A vizsgált két képzés hallgatói létszáma évfolyamonként 500–650 fő. A közoktatásban egy évfolyam ennek az ötöde, egy oktatott csoport körülbelül az 1/25-e. Az informatikaoktatás tartalmi megszervezése – a tanterv és a kimeneti követelmények – szintjén, azaz országosan, egy évfolyam 100 000 fő, a vizsgált képzések létszámának 200-szorosa. Az felsőoktatás szerepe és struktúrája a közoktatástól eltérő, a részletes elemzést a X/8. melléklet tartalmazza. Az egyetemi képzés szerepe 150 év alatt egyének kiváltságától a tömegképzésig [151] [152] változott. Ezzel együtt az oktatói szerep változása is jelentős volt, a személyes, éveken át tartó mestertanítvány kapcsolat helyett a félévente változó hallgatói–oktató kapcsolat a jellemző és a tanítási órák száma is megnövekedett. A magyar egyetemeket lassan nem utódneveléssel is foglalkozó kutatók, hanem kutatással is foglalkozó oktatók jellemzik. Az ilyen típusú oktatáshoz már komoly pedagógiai, andragógiai [153] jártasság lenne szükséges. Ennek tetejébe, a pedagógiai

paradigmák is gyökeresen átalakultak. A modern pedagógia a felsőoktatásban is a kompetencia alapú [154], tömegesen egyéni (személyre szabott) képzés felé tart.

7.3.4 Az oktatási módszerek hatékonysága

Kutatásom során nagyon sok előadáson, gyakorlaton és laborfoglalkozáson voltam bent, figyeltem a tananyag mellett az előadók módszereit, a hallgatók reakcióit, véleményeket, összehasonlításokat más előadásokkal. Esettanulmányok (XI/1) sora alapozza meg tapasztalataimat.

Az egyetemeken a tudás átadásának az alapja azelőadás (XI/2). A tanár oktatói elismerése az, hogy előadást tarthat, az előadás minőségének mutatója – ha önkéntes a részvétel – a hallgatók száma. Az előadó az „adó”, a hallgatók tömege a „vevő”. Az előadás során elhangzó kérdések sokszor költőiek. A BME-n az előadót a végén megtapsolják. Bár az előadás a tudás átadásának fő terepe, valójában a jó előadás élményt, többnyire tanulni való feladatokat ad.

A hallgató, ha nem kötelező az előadás, és van ideje, akkor az élmény miatt vesz részt szívesen egy előadáson vagy azért, mert a tanulnivalókról csak helyben tájékozódhat. Mivel az előadáson nincs valódi interakció, a hallgatók egy részének az előadás figyelése háttérben futó folyamat az éppen aktuális egyéb teendőikre koncentrálnak.

Egy 90 perces előadásból, akik figyelnek, azoknak is csak az eleje és a vége marad meg. A közvetlen tapasztalatból, a közoktatási tanóra felépítéséből, a konferenciák előadásainak szervezéséből, a TED előadások szabálya és szakmai ajánlások alapján kimondható, hogy egy előadás legfeljebb (18-) 20 perc lehet. Legfeljebb ennyi idő fordítható egy tananyag $LAU^{L}1$ fázisára, mert ez idő után a „vevő” inaktív.

A látogatott egyetemi előadásokra nem jellemző, de a fentiekből következik, hogy egy előadáson legfeljebb 20 percenként szünetet kell tartani, amelyben a hallgatók feldolgozzák az addig hallottakat, azaz lezajlik a $LAU^{L}2$ fázis. Ez az előadási szünet lehet egy közbeiktatott feladat, amit önállóan meg kell oldani, néhány tesztkérdés, egymás közötti konzultáció, visszakérdezési lehetőség… miközben a hallgató befogadás helyett az ismereteket feldolgozza. Javasolt a 45 perces előadást 2, a 90 perces előadást 4 egységre bontani.

Mivel az előadás egyszeri, minden előadást rögzíteni kellene, a rögzített anyagot tematikus egységekre bontva közzé kellene tenni. Egy egység legfeljebb 20 perces, de, ha a tematika lehetővé teszi, ezt 3–5 perces részekre célszerű tovább bontani, hogy a tanulás későbbi fázisaiban a kérdéses pontok gyorsan elérhetők legyenek.

Az előadás tanulás/tanítási hatékonysága nem csak az előadás szervezésén, felépítésén, az előadó kvalitásain és a tananyagon múlik, hanem a hallgatón is. Ebben azonban a hallgató viselkedése (pl. játszik, beszélget) nem ok, hanem következmény, visszajelzés arra vonatkozóan, hogy valami nem működik. A tanulás/tanítási hatékonyság, azaz a hasznosított/leadott ismeret aránya nagymértékben függ a hallgató korábbi ismereteitől is. Az odafigyelő és jegyzetelő hallgató számára az új ismeret lehet azonnal hasznosítható (└1A), nem csak a hallottakat jegyzi meg, hanem azt is, hogy hol tervezi felhasználni. De lehet, hogy a hallottak teljesen újak, a jegyzetelés lényege a pontos rögzítés és a megértés, megtanulás későbbi időre halasztása (└1P). Vagy, a hallottak megértése mellett a jegyzetben megjelennek kérdőjelek is, mert az önmagában érthető ismerethez nem kapcsolódik hasznosítási cél (└1M). Az aktív tanulónak (└1A) beindulhat a fantáziája, ötleteit megosztja, ezért beszélget; a passzív tanuló (└1P), ha a pontos rögzítés nem sikerül, akkor feladja, mert javítani sem tudja és így már jövőbeli haszon sem látszik. Az előadás hatékonysága a hallgató számára a további fázisokban (└2-3), szükséges tanulás mennyiségével értelmezhető. Az előadáson való aktivitása (részvétele) ennek becsült mértékétől és az ismeretek más forrásból való elérhetőségétől függ.

A fentiekből következik, hogy az előadások mennyiségét célszerű minimális szinten tartani, mellette más ismeretforrásokat biztosítani. Az előadás után a hallgatóknak egyedi tanulási feladatai vannak.

A gyakorlat és labor (XI/3) a két intézményben eltérő értelmű, de a megvalósítás részben hasonló. Legfőbb közös vonás a *beugró*, az óra elején írt kis dolgozat. Ez leginkább a közoktatásban irodalomból ismert olvasmányellenőrzéshez hasonlít, az előadás tananyagának ismeretéből mintavételes ellenőrzés. Hibának tartom, hogy a beugró – bár eltérő módon, de végső soron – a gyakorlaton való részvételt, teljesítményt előzetesen minősíti. Ugyanakkor fontos, hogy a gyakorlat elején legyen valami „bemelegítő", összefoglaló a gyakorlat alapjául szolgáló ismeretekből. A hiányokat nem a foglalkozás elején kellene minősíteni, hanem a végén, esetleg egy „*kiugró*" íratásával, az órai munka ellenőrzésével.

A beugró helyett *Házi feladat* és ennek ellenőrzése is megoldás lehetne az előzetes felkészültség ellenőrzésére. Az ELTE-n ez megvalósítható, a BME-n a feladatok (beugró) központi kiadása mellett nincs rá tér. A házi feladat értékelésben mindkét képzésben a legnagyobb gondot a másolt megoldások kezelése jelenti. Ezt figyelembe véve, hasznosnak gondolom:

- kérdéssor kiadását, amire rövid, esszé jellegű válaszokkal lehet a gyakorlatra készülni, ennek előzetes bekérése vagy az óra elején 1-1 válasz helyben leírása értékelhető.

– tesztfeladatok, feladatbank kiadását, a gyakorlat elején (pl. 5 perc alatt 10%-ot) számonkérve a felkészültség gépi kiértékeléssel ellenőrizhető.

Mindkét esetben biztosítani kell, hogy a válaszok helyességét a hallgató önállóan tudja ellenőrizni, ha problémája, kérdése van, tudjon segítséget kérni. Az értékelésnél az ismeret L1, esetleg L2-3 fázisát mérjük, ami a befogadás, a passzív rögzítés tényének megállapításáról szól, nem az alkotásról nem kreativitásról. A számonkérés hitelességét megfelelő környezet megteremtésével, a kiadott kérdések szétosztásával, sorrendjének változtatásával lehet biztosítani.

A laborok, gépes gyakorlatok fő funkciója az ismeret L2 és L5a-5b fejlesztése, az „elméleti" gyakorlatokon a L3, L5b szintűek a feladatok. A L2 és L3 során segítséggel, a L5a, L5b esetében önállóan kell az adott ismeretet alkalmazni, de mindenképpen gondolkodva, felfedezve, értelmezve a tananyagot. Sokszor tapasztaltam, de hiba, ha az oktató újra elmagyarázza a tananyagot, az viszont lehetséges, hogy a laboron, gyakorlaton új ismeret is előjön, amire csoport szinten érdemes reagálni. Azt tapasztaltam, hogy nem segíti az önálló gondolkodást a feladatok közös megoldása, sem a tanulók közötti kooperáció, mert a hallgatók többsége ilyenkor csak követi a gondolatmenetet. Másrészt segít a megoldások megbeszélése és nehézségek esetén a közös tervezés. A megoldásban elakadó tanulónak segítséget jelent a lehetőség a probléma hangos kifejtésére, segítséget jelent a lehetséges hibákról, illetve a megoldási lehetőségekről egy lista. A probléma oktatói megoldása csak a feladat elkészítésében jelent segítséget, a tanulásban alig (a minta adása az előadás, a videó része).

Mivel a közoktatásban lényegében gépes gyakorlat (labor) keretében történik az informatikaoktatás, az előadást, mint új tananyag ismertetését is beleértve, a tapasztalatok alapján a közoktatásban is az alábbi módszerek használatára törekszem:

– Ha az órán új ismeret megtanulása szükséges, akkor előre jelzem, hogy melyik ponton, részfeladatnál számítsanak a tanulók az egyéni munka felfüggesztésére.
– Aki nagyon elől jár a megoldásokkal, annak magyarázat helyett inkább forrást mutatok, így önállóan bővítheti a tudását.
– Aki lemaradt, ahhoz megkérem az előrébb járót, hogy ő mondja el. Ezzel az is gyakorol, tanul, aki a másiknak segít. Ha lehet, meg is hallgatom a magyarázatot.
– Akinek egyéni problémája van – nem működik, nem úgy működik a programja vagy az alkalmazás, ahogy elvárja –, annak el kell mondania pontosan a problémát. Ennek célja, hogy megtanuljon jól kérdezni, ami sokszor a választ is tartalmazza. A kérdésére nem a választ mondom meg, hanem a forrást, ahol utánanézhet vagy kiemelem a kérdésből a keresési kulcsszót.

- A feladat értelmezésével kapcsolatos probléma jellemzően:
  - a feladat egy részét nem veszi figyelembe a tanuló, a segítség: „Mutasd a feladatot!”.
  - egy-egy szó, fogalom megértése okoz gondot, vagy a mondat tagolása nem egyértelmű, ilyenkor az értelmezések diszkussziójában segítek. Az ellentmondások kiszűrése után a lehetséges értelmezések mindegyikének kipróbálását javaslom, mert ez biztosítja a megfelelő (pozitív és negatív) tapasztalatot.
- A feladatokat egyénileg kell megoldani, de a feladat értelmezését meg lehet beszélni. A kész megoldásokat célszerű legalább egy társ megoldásával összehasonlítani, az eltéréseket megbeszélni.
- Másolni tilos. Nem csak a copy-paste tiltott, képernyőről, dokumentumból másolás és a diktálás is. Egy részmegoldás megtekintése és leírása (esetleg reprodukálása) között legyen legalább annyi idő, amennyi idő alatt az adott részt el lehet mondani.
- Saját kódot sem szabad másolni. Ha egy gyakorlaton 50-szer kell for-ciklust írni, akkor mind az 50 esetben végig kell írni.

A programozás oktatásának és tanulásának módszerében nehéz, hogy egyszerre háromféle tanulási módszert kell alkalmazni.

1. Az elveket, gondolkodási módszereket, sémákat hosszú távon, készség szinten kell tudni, ehhez változatos alkalmazási helyzetek szükségesek. Legjobb szóban, személyes (humán-humán) interakciókban fejleszteni: mondja el, hogyan érti, mondja el a megoldását, indokolja a megoldást. [25, 28, 32, 33]
2. A programozási nyelv vagy alkalmazás alapszavait, szintaktikai szabályait be kell magolni, sőt, az az igazi, ha nem fejben, hanem „az ujjainkban van” a tudás, ami sok gyakorlással, ismétléssel érhető el. [25, 28]
3. Végül, ahhoz, hogy egy feladatot meg tudjunk oldani, a programban használt változóneveket, függvényneveket, az alkalmazásban szereplő fogalmakat, összefüggéseket a munka idejére fejben kell tartani. [25, 32]

A hatékony tanuláshoz fontos, hogy a tanuló is tisztában legyen azzal, hogy mit hogyan kell tudnia. Oktatói részről folyamatos önkontrol kell arra vonatkozóan, hogy mit akar megtanítani, melyik területet akarja fejleszteni. A két egyetemi képzésben (is) hangsúlyt kell fektetni arra, hogy programozási elveket, modellezést (informatikát) tanítunk (1. pont, gondolkodási készség fejlesztése), valamint a nyelv csak eszköz (2. pont, rutin kialakítása). Informatikaoktatásban ezzel azonos szemléletű problémamegoldás (1. pont), illetve az alkalmazás oktatása (2. pont).

7.3.4.1 Projekt

Mindkét megfigyelt tárgyban van egyéni projektfeladat, „*nagy beadandó*” (XI/4) néven. Ezek a tanultak L5b-5c ismereteit igénylik, azaz a projekt során nem kell új tananyagot megtanulni, a tanultak kreatív alkalmazása a feladat. A projektfeladat elkészítését konzultációk segítik.

Elképzelhető a tananyag projektalapú feldolgozása, amelyben a beadandók elkészítése lehetne a középpontban. Ekkor a gyakorlatok és laborok helyébe konzultációs lehetőséggel kiegészített megoldásra allokált hely és idő kerül. A konzultáción a témát a tanulói kérdések is befolyásolják, a részvétel nem kötelező, az értékelés módja is módosul.

Távoktatási környezetben az oktatás projekt alapú megszervezése hatékonyabbnak tűnik.

7.3.5 Mérés-értékelés

A mérés és az értékelés az oktatás elválaszthatatlan része. Az egyetemi informatikaoktatást elemezve azt tapasztaltam, hogy ezen a területen komoly problémák vannak. Miközben szakmailag sok mindent tanultam, „kinyílt a világ”, a mérés-értékelés területén a megállapításaim inkább a kellemetlen tapasztalatok elkerülését szolgálja. Számos esetet és ezek elemzését tartalmazza a XII. melléklet. Az informatikaoktatás módszertani kérdései szempontjából a hol, kivel, mikor történt kérdések lényegtelenek. A tanulságokat, a mérés-értékelés során alkalmazandó szabályokat foglalom össze [11 p:392–417, 155].

A tanuló értékelése az értékelő személye alapján lehet

- *önértékelés*: a tanuló saját munkáját értékeli;
- *oktatói értékelés*: a tanulót saját tanára értékeli. A tanítás perszonális kapcsolatából származó szubjektív tapasztalatokat és egyéni jellegeket is figyelembe tudja venni.
- *társértékelés*: a tanulót tanulótársai értékelik. Az oktatói értékeléstől két szempontból tér el: nincs alá-fölé rendeltségi viszony, illetve a társ szakmai hitelessége nem biztosított.
- *központi értékelés*, amelyben a tanulót külső mérőeszköz segítségével értékelünk, a mérés összeállítója, az értékelés leírása az oktatótól független.

Az értékelés funkciója alapján lehet diagnosztikus, formatív vagy szummatív.

- *Diagnosztikus* értékelés szükséges egy-egy téma tanításának elején, az értékelés a témához szükséges LAU-ok fázisára vonatkozik. Jó, ha a csoport tagjai önállóan képesek elvégezni és jelzik, ha egy LAU-ban elvárt fázistól jelentősen eltér a tudásuk. Emellett, ha egy tanuló elakad egy feladat megoldásában, akkor a segítséget megelőzően diagnosztikus értékeléssel határozzuk meg, a probléma okát.

- Az informatika – programozás – művelése gyakorlati tevékenységként nyilvánul meg. A tanuló tevékenységét figyelve, szinte folyamatosan *formatív* értékelést kell végeznie az oktatónak. Minősíti a készülő megoldást, bíztatja a tanulót, ha jó irányba halad, folyamatos jelzést ad a munka állapotáról. A követelmények nagyon pontos ismerete szükséges ahhoz, hogy a tanuló önmagát formatívan értékelje, ugyanakkor gépi ellenőrzéssel az értékelés támogatható.
- A *szummatív* értékelés elvileg a téma lezárásakor, az elért eredmények rögzítésére szolgál. A LAU-modellben a „téma lezárása" értelmezhetetlen, a szummatív értékelés eredménye pillanatnyi állapotot tükröz.
  - A tanítási gyakorlatban „próba" szummatív értékelések adhatják az alapját a diagnosztikus értékelésnek.
  - A tanterv szerint meghatározott időpontban történik a mérés, ami állapotfelmérés.
  - Minden szummatív értékelést célszerű diagnosztikus értékeléssel is kiegészíteni (a továbbhaladás szempontjából is értékelni a számszerű eredményt). Értékelést követő fejlesztés, fejlődés után – ha a tanuló számára a téma nem zárult le – újra kell értékelni.

A közoktatásban fejlesztendő kompetenciaterület a diagnosztikus és formatív önértékelés, tanmenetben tervezett a diagnosztikus oktatói értékelés, és tanítási módszer része a formatív oktatói értékelés. A szummatív értékelés lehet oktatói és központi is.

A vizsgált egyetemi tantárgyak esetén (XII/1) elvárás az önértékelés, de ehhez jellemzően hiányosak a mérő eszközök, így a minősítés sem teljes. A hagyományos mester-tanítvány kapcsolatban az oktatói értékelés dominált, de a tömegképzésben az oktató és hallgató között a perszonális kapcsolat gyenge: előadó-hallgató esetén lényegében nincs, csoportot oktató, illetve demonstrátor és hallgató között időben rövidtávú. A diagnosztikus és formatív értékelés sok esetben nem feladata az oktatónak. Tipikus, hogy a demonstrátor, azért van jelen a gyakorlaton, hogy az önálló munka során felmerülő kérdésekre válaszoljon, amiről esetenként átvált előadásba. A szummatív értékelés a BME mérnökinformatikus képzésben központi, az ELTE programtervező informatikus képzésben van oktatói és központi értékelés is.

Az informatika tanulásában – és ezért az oktatásában is – a diagnosztikus és formatív értékelésnek van nagy szerepe, az fejleszti a kreativitást, az informatikai gondolkodást, a rendszerben gondolkodást és az önértékelést is.

A továbbhaladás és a minősítés szempontjából lényeges, rögzített értékelés szummatív. Sajnos sok esetben ez még úgy is igaz, hogy nem a tudás számít, hanem a megszerzett jegy: „csak

görbüljön”. A minőségi informatikaoktatáshoz olyan értékelési környezet kell, ahol a szummatív értékelés pontos, a valódi tudást tükrözi. Ez a mérés-értékelési rendszer megfelelő kialakításával, minőségbiztosítási szempontok érvényesítésével valósítható meg. A tapasztaltak alapján (XII/2, XII/3) megfogalmazott mérési elvek:

- A mérést megelőzően pontosan kell megadni, hogy mi lesz a mérés tárgya. Pontosságba nem csak az ismeret vagy készség megnevezése, de az elvárt tudás szintje (LAU-fázisa) is beleértendő.
- A mérőeszközt – feladatsort – a mérés tárgyának megfelelően kell elkészíteni. értékelési útmutató kell hozzá, amiben meg kell adni az értékelés szempontjait, ezek mérési helyeit, a hozzá rendelt – esetleg súlyozott – értékeket.
  - Egyetlen csoport esetén, ha csak néhány szempont van nem szükséges leírni, elég szóban közölni az értékelés módját.
  - A mérés helye lehet a teljes munka is (például „nincs egyetlen üres bekezdés sem”).
  - A feladatban nem mindig szerepel az értékelés szempontja (például: „minta alapján…”).
- A feladatsort értelmezési lehetőségek és eltérő megoldási módszerek szempontjából ellenőrizni kell, biztosítani kell, hogy az értékelendő ismeretet, készséget mérje. Figyelni kell arra, hogy informatikából jellemző, hogy a megoldási módot/utat/gondolatot értékeljük a kapott eredményen keresztül!
- A feladatsor megoldását időtartamra is ellenőrizni kell. Amennyiben a mérés egy ismeret vagy készség…
  - L5a szintjére vonatkozik, akkor a konkrét feladatnak előre ismertnek kell lennie, a számonkérésre szánt idő a tanár megoldási idejének duplája legyen – A dolgozat írásakor ellenőrizni is kell a megoldást.
  - L5b szintjére vonatkozik, akkor a lehetséges feladattípusról kell tájékoztatni a tanulót, a megoldási idő a tanár megoldási idejének legalább háromszorosa [155] legyen. – A dolgozat írásakor ki kell találni, hogy melyik ismeret vagy készség felhasználása szükséges, meg kell oldani a feladatot és ellenőrizni is kell. A megoldáshoz szükséges időbe bele kell számolni, hogy a dolgozatot író nem gondolatolvasó, a feladat kiírójának szándékát fokozatosan érti meg, a megoldás közben hibázhat, a hiba javítása is idő.
  - L5c szintjére vonatkozik, akkor a megoldásra napokat kell biztosítani, mert a kreativitást, új alkotást igénylő problémamegoldás nem rutinszerű tevékenység.

Itt is látható, hogy L5c szintű tudás értékelésére jellemzően projektfeladat vagy élethelyzet alkalmas. Egy kötött idejű dolgozattal legfeljebb L5b szintű tudás mérhető, az idő-megszorításos mérések a „kapcsoltam” jellegű L5a szintű tudás mérésére alkalmasak.

- A mérés fizikai környezetét úgy kell kialakítani, hogy pontosan azok az eszközök legyenek elérhetők, amelyek használatára számítunk (jegyzet, internet, társak, tanári segítség).
  - Központilag készített feladatok esetén a méréskor a felügyelő nem segíthet.
  - Ha a feladat készítője a mérésben résztvevő összes tanulóval kapcsolatban van, akkor egyéni kérdést (közvetítve neki) megválaszolhat mindenkinek, de ez jelentősen rontja a mérés minőségét, mert a tanulókat a válasz különböző munkafázisban éri el, megzavarhatja, bosszanthatja (például, ha már megoldott részhez új szempontot kap).
  - A feladat értelmezésével kapcsolatos kérdést és minden felmerülő problémát célszerű a dolgozathoz mellékelni, ezt célszerű az értékeléskor figyelembe venni. Szükség esetén a mérést vagy egy részét érdemes megismételni.
  - A dolgozat ideje alatt hangosan feltett kérdés a többieknek küldött segítség, ennek megfelelően kell értékelni a kérdező és a többiek munkáját is.

Az értékelés (XII/4, XII/5, XII/6)

- Az értékelésnek pontosan kell jeleznie a hibák helyét és típusát (melyik értékelési szempontot nem teljesíti).
- Hasznos, ha a dolgozat értékelését – az értékelési útmutatás alapján – a dolgozat írója is elvégzi.
- Az értékelést a dolgozat írójával közölni kell. A dolgozat készítőjének célszerű a hibákat elemeznie, megtervezni – esetleg megbeszélni – a további teendőket, a megoldást javítani. Amennyiben az értékeléssel nem ért egyet, akkor az értékelővel konzultálni kell.
- Az értékelés lezárását követi a minősítés. Még előre ismert ponthatárok esetén is a minősítés során – szakmai indoklással – el lehet térni (például értékelhetetlen rész figyelmen kívül hagyása, eszközhiba figyelembevétele…).
- A mérésnek és az értékelésnek objektívnek kell lennie, a minősítés tartalmazhat szubjektív tényezőket. Fordítva, a mérés és értékelés során a szubjektív tényezők rontják az értékelés hitelességét, a minősítés kőbevésése figyelmen kívül hagyja a mérés pontatlanságát.

A szummatív mérés és értékelés a tanári munka legkevésbé kedvelt tevékenysége. A felügyelet is és az értékelés is sokszor monoton, személytelen, lényegében automatizálható, gépesíthető feladat. Ezért célszerű optimalizálni a feladatokat:

- A mérés helyszínén az illegális tevékenységet a lehetőségekhez mérten ki kell zárni, ami nem kizárható, annak detektálását célszerű biztosítani.
  - Ültetési távolság, táskák, telefonok távoli elhelyezése, csak kábeles kommunikációra képes eszköz használata, adatforgalom tiltása és logolása.
  - Felügyelet hátulról – jó lenne kamerás megfigyelés.
  - Fájlméret, készítő, mentési adatok figyelése, plágium ellenőrzés.
- A dolgozat formájánál szempont, hogy könnyen ellenőrizhető legyen.
  - Elektronikusan tárolt. Jól olvasható. Hiteles eredeti példány, másolata javítható.
  - Közvetlenül értékelhető (nem tömörített, korrektúrázható), „ránézésre" eldönthető, hogy helyes-e a megoldás vagy könnyen tesztelhető.
  - Esetleg automatikusan értékelhető: itemenként jó, hibás vagy egyedi, ahol az egyedi értékelést kell csak a tanárnak elbírálnia.
- Áttekinthető, visszakereshető, elemezhető értékelés
  - értékelő táblázat/űrlap használata.
  - számítógépen tárolt, a tanulóval megosztható az értékelés minden részlete
  - számítógépes előértékeléssel feltölthető.

A mérés-értékelés jelentős része megoldható lenne mesterséges intelligencia asszisztenssel. Amíg az nincs, addig is, a megoldást főleg a számítógépes részmegoldások jelentik.

A közoktatásban elegendő ötfokozatú skálán mérni, ami egyes dolgozatoknál (akár témazáróknál is) lehetővé teszi nagy egységek értékelését. (Például: „értelemszerűen használ bekezdésformázást", „statisztikai függvények használata helyes".) Kis gyakorlattal akár egy érettségi feladatsor is ránézésre minősíthető, azonban a fejlesztő értékeléshez ez kevés. Másrészt, az értékelés akkor is fejlesztő, ha a tanuló végzi, ezzel az önértékelése is és önállósága is fejlődik. Kontrollált önértékelés vagy társértékelés – az értékelés során a tanuló kérdezhet is – részben helyettesítheti a MI asszisztenst és fejlesztő is. Ezért ezt használom az érettségi felkészítésben.

Az egyetemi informatikusképzés szakmai alapozó tárgyait egy középiskolai csoport 30–60-szorosa tanulja. Ezeknek a tárgyaknak a központi méréseit úgy kell megszervezni, mint a közoktatásban a központi méréseket, például az érettségit. Célszerű bemért feladatokból bankot készíteni, ezt folyamatosan fejleszteni, javítani. Az értékelést standardizálni kell, hogy a különböző időpontokban vizsgázók tudása azonos szempontok és súlyozások alapján legyen értékelve. A dolgozatok értékelése ilyen méretekben óriási munkaterhelés, azért kiemelten érdemes gépesíteni, automatizálni a L5a szint mérését, és minél nagyobb gépi támogatást szervezni a L5b szintet mérő feladatok értékeléséhez.

A mérés-értékelés leírásából is kitűnik, hogy nem lehet MI asszisztensre bízni az egyedi megoldások elbírálását. Ez igaz a projektek értékelésére is, amelyekben mindenkinek egyedi megoldása van. A formai megfelelés ellenőrzésén túl, a L5c, alkotó kreativitást igénylő feladatok értékelése az oktató vagy társak feladata.

7.3.5.1 Zavarok a szerepek körül

A feladatok, gyakorlatok megoldása során gyakori, hogy a teljesítést a feladat elkészülte jelenti. Tudatosítani kell – rákérdezéssel, beszámoltatással –, hogy a konkrét feladat mindig csak eszköz az informatikai (programozási) ismeret, készség fejlesztéséhez, gyakorlásához. A feladat témája, a tananyag „körítése", amelynek a problémafelvetés mellett sokszor célja a motiváció növelése, a (szituációs) játékhelyzet megteremtése.

A diagnosztikus és formatív értékelés közös jellemzője, hogy szubjektív tényezői is vannak, jelentős az empátia szerepe. Fejlesztő hatása nem csak a pontos helyzetfelismerésnek és tanulási útvonal meghatározásának van (ebben kell profinak lenni az oktatónak), hanem a motiválásnak, lelkesítésnek is. Ezért a társértékelésben a kortárs (korrepetitor, demonstrátor, diáktárs) hivatalosan megadott kapcsolatán túl, a baráti kör reflexiója is meghatározó lehet [156].

A szummatív mérésnek és értékelésnek objektívnek kell lennie. Ez határozott szerepváltást igényel az oktató részéről, amelyet a tanulóval is el kell fogadtatni. Az óra előtt még empatikus, segítőkész oktatóból a dolgozat idejére rideg felügyelővé kell válni. Ez a feltétele annak, hogy az oktató ne a saját gondolatait értékelje, hanem a tanuló önálló munkáját. A minőségbiztosítás szempontjából – és lelkileg is – sokat segít, ha a szummatív mérést a tanítástól külön szervezik, a tanítástól független standardok és lebonyolítók bevetésével.

A szummatív értékelésen alapuló minősítés esetenként a meghirdetetthez képest módosulhat, de ilyenkor is csak a szakmai szempontok vehetők figyelembe. Ide tartozik a reklamációk elbírálása is. Az értékelést követően a hallgató önállóan vagy társai, oktatója segítségével diagnosztikusan is értékeli a teljesítményt, elemzi az értékelést. Ez a tevékenység szubjektivitást igényel. A minősítésnek viszont objektívnek kell lennie. Amennyiben az oktató nem tud váltani a két szerep között, célszerű külső, hitelesített szakértelemmel rendelkező személyt felkérni az elbírálásra. A pedagógusi és egyetemi oktatói pályán is a kiégés egyik tipikus oka, ha az oktató szakmai hitelességét az együttérzésének elvárásával támadják.

A demonstrátorok és a gyakorló tanárok az oktató-értékelő szerepváltás szempontjából kiemelten veszélyeztetettek. Könnyebben tudnak érvényesülni a tanulásban társként, segítőként, ugyanakkor gyengébb a szakmai hitelességük, amiben nem csak a tudás számít, hanem a minőségi követelmények is.

7.3.6 A programozási nyelv és alternatív platformok

A közoktatásban és az egyetemi képzésben is állandó téma, hogy a közel azonos célú szoftvereszközök közül melyiket tanítsuk, melyikkel tanítsunk. A közoktatásban az operációs rendszer, az office programcsomag, a csoportmunka szoftver, az adminisztrációban használt eszközök, a grafikai, weblapkészítő és médiaszerkesztő alkalmazások mellett jelentős hangsúlyt kap a programozási nyelv és fejlesztő környezet kiválasztása. A felsőoktatásban a hallgatók a billentyűzetkiosztásra érzékenyebbek[25], az oktatók sok tekintetben igyekeznek multiplatformot biztosítani, elfogadni az általuk használttól eltérő alkalmazásokat is, de jellemző, hogy egyet használnak. A profilnak – informatikusképzés – megfelelően a képzés tervezésekor központi kérdés a tanítandó programozási nyelv kiválasztása. Pontosabban, az, hogy mely nyelveket mikor tanítsák, melyek legyenek a mindenki számára kötelezően tanult nyelvek. Mivel a kérdés a közoktatásban rendszeresen jelen van – országos és helyi szinten is –, feszültségforrás az oktatáskutatók, a tananyagkészítők között és a munkaközösségeken belül is, ezért elemeztem az egyetemek válaszait a kérdésre. A XIII. mellékletből itt csak a fő választási szempontokat mutatom be, ebből levezetve, hogy a közoktatásban mely szempontok mérvadóak.

A BME VIK-en C az első nyelv. Bár nehéz nyelv, az adatorientált programozásoktatásnak a C felel meg leginkább. A C++ nyelvre áttérésnél kérdés, hogy adat- vagy objektum-orientált programozással kezdjenek, valamint a villamosmérnökök is azt tanulják-e, mint a mérnökinformatikusok. A 2018-ban indult BProf, üzemmérnökinformatikus képzésen Python az első nyelv. Itt nem a gépközeli kódolás az elsődleges, hanem az alkalmazásfejlesztés.

Az ELTE IK-n az algoritmus-orientált módszeren van a hangsúly, a nyelv kevésbé fontos. A programozás alapismereteinek oktatásához három nyelvet használ: matematikait a specifikációhoz, struktogramot a modellezéshez és C++ alapokat a kódoláshoz. Ezzel párhuzamosan tanítanak az Imperatív programozás tantárgyban C-t és Pythont (matematikához), a Funkcionális programozás tantárgyban Haskellt, a Számítógépes rendszerek tárgyban PowerShellt és Bash scriptet. A nyelvek egy részét a hallgatók keverik, másik részét hamar elfelejtik. A tanított nyelvek sokféleségének oka nem a módszertan, hanem az oktatók egyéni preferenciája.

Az informatikusnak – programozónak, programtervezőnek, mérnökinformatikusnak, gazdasági informatikusnak, … informatikatanárnak – képesnek kell lennie arra, hogy a munkájához szükséges programozási nyelvet a felhasználás igényének mélységében megtanulja. Ezt alapozzák meg azok a nyelvek, amelyeket a képzése során tanult. Az ELTE-n a programozási nyelv

[25] A közoktatásban a magyar nyelv és a magyar billentyűzet-kiosztás használata előírás. A felsőoktatásban jelentős a külföldi hallgatók aránya, fokozott az igény az angol billentyűzetre. A váltogatással van probléma.

egy eszköz, a nyelv megismerése azt jelenti, hogy a használatát ismeri meg a programtervező; ezzel szemben a BME VIK-en a nyelv termék is, szükséges a „hogyan működik” vizsgálata is.

Költői kérdés, hogy több nyelv felszínes ismerete vagy egy nyelv mélyebb ismerete segíti-e jobban a későbbiekben egy tetszőleges nyelv tetszőleges mélységű megismerését. Az „egyszerre egy nyelvet tanulj” elvet támasztja alá Robert C. Martin [157 p:41] programozóknak szóló tanácsa, hogy munka előtt KATA-t kell írni, amely az aznapi munkához szükséges tudáscsomagot előhozza. Oktatóknál is megfigyelhető az egynyelvűség. Például a MOOC tanfolyamon látható, hogy David Malan C-ben „otthon van”, a Pythont csak ismeri[26].

A Columbia University, Harvard College, University of Cambridge tantervében[27] a programozás elméleti alapjainak megismerése előtt ajánlják valamelyik nyelv megtanulását, vagy az első szemeszter elején pár órában tanítanak Pythont vagy Scratch-et. Ezt követi az oktatási célnak megfelelő nyelv tanulása, használata.

Egyes diákjaim jelzései alapján, de saját tapasztalatom is az, hogy gyors váltásban különböző nyelveken programozni lehet, de mindenképp megterhelő. Van, akinek a plusz terhelés „nem számít”, másokat megakadályoz a gondolkodásban, ami rontja a munka hatékonyságát.

A közoktatásban a tantervek a fentiekhez hasonló megfontolást tesznek lehetővé. A felsőtagozat képzésében bevezető nyelvként egy blokk nyelvet jelölnek meg és megjelenik mellette a robotok programozása is. Korábban a Logo nyelv szerepelt ugyanerre a korosztályra. Szerepét tekintve ide sorolható a Python is, amely Logo helyett is használható és blokk-nyelvből is lehet Python kódot készíteni. A középiskolai oktatás programozási nyelvének kiválasztásához a megtanítandó algoritmusok és adatstruktúrák adnak útmutatást. Olyan programozási nyelvet érdemes választani, amiben a „kezdő szintű” program konzolos, a nyelv jól lehatárolható elemi részében (subset) a tanítandó vezérlési struktúrák megtalálhatók és erősen, statikusan típusos.

A programozási nyelv és az alkalmazástípusok esetén is, a választást erősen befolyásolja a tanár tudása, rutinja, valamint a fejlesztőkörnyezetek és alkalmazások minősége. Senkitől nem várható el, hogy egyik óráról a másikra nyelvet, fejlesztőkörnyezetet, alkalmazást váltson, de elvárható, hogy élethosszig bővítse tudását, módosítson szokásain, rutinjain.

---

[26] C: https://www.youtube.com/watch?v=u-kH-5JJSgU [2021.10.30]
Python: https://www.youtube.com/watch?v=hnDU1G9hWqU [2021.10.30]

[27] Columbia: http://www.cs.columbia.edu/~bauer/cs3101-1/ [2021.10.30];
Harvard: https://online-learning.harvard.edu/course/cs50-introduction-computer-science [2021.10.30], https://www.youtube.com/watch?v=u-kH-5JJSgU [2021.10.30];
Cambridge: https://www.cst.cam.ac.uk/admissions/undergraduate/faqs, https://www.cl.cam.ac.uk/teaching/1920/cst.pdf [2021.10.30]

8 Az informatika (programozás) tanítása – eredmények felhasználása

A kutatásom során elemeztem a közoktatáson belül az informatikatantárgy szerepét, tartalmát; az oktatás módszertani kérdéseit (4–7). Ezen belül megvizsgáltam két nagy informatikusképző oktatási rendszerét, programozás és néhány más tantárgy tartalmát, felépítését, az oktatás módszertanát; megvizsgáltam, mitől függ a tanulás sikeressége, melyek a hátráltató tényezők.

A tapasztalatokat saját oktatási gyakorlatomban felhasználtam, alkalmaztam az elmélettel is alátámasztott „jó gyakorlatokat". Az kutatás eredményeit pedagógusként a tanítási gyakorlatomba is beépítettem, az informatikaoktatás módszereihez hozzárendeltem a pedagógiai eszköztárt, módszereket, tanítási formákat.

Az informatika (programozás) gyakorlati tárgy. De nem úgy, ahogy sokan látják, nem a gép nyomogatása, eszközkezelés miatt gyakorlati, hanem a gondolkodás, értelmezés, elemzés, modellalkotás gyakorlata az, amit a tantárgy tanulása során meg kellene tanuljon a tanuló. Az informatikaoktatás legfontosabb és legnehezebb feladata a gondolkodás megtanítása. Az Informatika (4.7) meghatározásából az oktatásban az **informatikai gondolkodás** az, ami köré a specifikáció további részei felfűzhetők.

A gondolkodáshoz szükség van ismeretekre, adatokra, amelyeket megtanulunk. A tanulás történhet memorizálással, vagy gondolkodásunk eredményeként. Az iskolai környezet jellemzője, hogy adott időre meg kell tanulni adott tananyagokat. Erre a memorizálás alkalmas inkább, mert az – egy kis önismerettel – bemérhető, hogy mennyi idő alatt, hányszor olvasva, hányszor ismételve teljesíthető. A memorizálással – mint passzív ismeretszerzéssel – azonban együtt jár a felejtés [24, 25]. A vizsgára, az adott időpontra szól a tudás. Ezzel szemben az informatikai gondolkodás során az agy nem befogad, hanem alkot, próbálkozik, eseteket elemez [37, 32]. Egy-egy probléma megoldásának az ideje nehezen határozható meg, ami a szűk határidő miatt elég riasztó [33]. Egyes problématípusok gyakori megoldásával a hozzá szükséges gondolkodás rutinná válik, amitől felgyorsul a megoldás kitalálása.

A tanítási gyakorlatom egyik alapelve, hogy *a kreatív, értelmező, informatikai gondolkodás – computational thinking – cselekvő tevékenység, az ember veleszületett képessége. A tanítás/tanulás során ezt a képességet felfedezzük – azaz tudatosítjuk, hogy létezik – és fejlesztjük.*

Az informatika oktatás (6. fejezet) mindennapi gyakorlata során az informatikát tanító tantárgy – Informatika, Digitális kultúra – tematikájába (5. fejezet) integráltam az informatikát (4. fejezet). Az egyetemi és közoktatási képzések (7. fejezet) elemzése alapján dolgoztam ki az oktatáshoz és készségfejlesztéshez megfelelő feladatokat, méréseket, tanítási gyakorlatot.

8.1 Alkalmazott tanóraszervezési módszerek

Tanítási órán hagyományos értelemben vett előadást nem tartok. Nincs előre elkészített prezentációm sem. Az előadás helyett általában mesélek, beszélgetünk, bemutatok egy megoldást, vagy elemzünk egy megoldást. A diákoktól elvárom, hogy egy-egy kulcsszót megjegyezzenek, a témának önállóan utána olvassanak, de a tanórán a kapcsolódó ismeretek felelevenítésén, a fogalmak értelmezésén van a hangsúly.

8.1.1 Informatikai eszközök használata és Információs technológiák témakör

Az Informatikai eszközök használata témakörhöz kézbe adok szétszerelhető hardver eszközöket. A kettes számrendszert fiúknak „bináris fekvőtámasszal" tanítom – az egyes helyiértékek a diákok, 0 = karhajlítás, 1 = nyújtva –, lányoknak guggolással, házi feladatként az ujjakon számolással. A dolgozat előtt több héttel (félidőben) kiadom a lehetséges kérdéseket, amelyekre a válaszokat egyéni projektben, önállóan keresik meg a diákok. Ajánlott források: internet, könyvtár, ismerősök, osztálytársak, saját tapasztalat – igyekezzen ellenőrizni a válasz helyességét több forrásból. Kérésre egy-egy választ előre értékelek, de a dolgozatban saját megfogalmazásokat várok. A tanórákon az ismeretek keretét igyekszem megadni, szerepelnek a kulcsszavak, ehhez kapcsolom a diákok saját addigi tapasztalatát. Jellemző tanórai kérdés: „Mi lehet a hiba, ha…?", ami például folytatódhat úgy, hogy „nincs hang". Nincs projektfeladat egy-egy hardver bemutatására, mert az csak adatok összeollózása, de lehet projektfeladat a hibalehetőségek feltérképezése.

Az Információs technológiák témakörhöz van jegyzet, de az elméleti tananyag órai megbeszélését bármilyen forrásból lehet tanulni. A fogalmak megnevezése és a működés elve, logikája van itt is a középpontban, de a dolgozatban csak a fogalmak ismeretét mérem. Az UTP kábel szerelése, hálózat tesztelése, alkalmazások konfigurálása és közös használata gyakorlati feladat. A dolgozat egy ad hoc csoport munkaterületeinek a létrehozása, használata, feladatok elosztása – mindezt úgy, hogy online bármi használható, de beszélni nem lehet.

8.1.2 Digitális írástudás témakör

A Digitális írástudás témakör egyik alkalmazásához vagy új megoldási mód ismertetéséhez sincs prezentációm. Az alkalmazás funkciójának tárgyalása beszélgetés formájában történik, a használatát vezetett gyakorlatként tanítom. Ez azt jelenti, hogy mindenki a saját eszközén nézi, csinálja azt, amit mondok. Ha valaki valamit nem ért, annak segítek, bevárunk mindenkit. Szándékos, hogy nem mutatom meg előre a menüt, nem adok mellé filmet… Célom az, hogy ne

másolja a diák a tevékenységemet, hanem értelmezze, amit mondok és fedezze fel az alkalmazást. Ezzel három dolgot kapcsolok össze: a látványt – amit maga előtt lát –, a kapcsolódó fogalmakat és a cselekvést. Például: „Kapcsold be a szűrőt, a Kezdőlap szalag jobb szélénél található tölcsérszerű ikonnal! Megjelent az első sor celláiban egy-egy háromszög, lenyíló menüt jelölő nyíl.” A szóbeli, szöveges instrukció előnye, hogy közben nem kell vizuálisan váltania a tanulónak. Nem mintát követ, hanem értelmezi a hallottakat. Kiemelten előny a mintakövető tanuláshoz képest, hogy a megoldás felfedezése közben a kapcsolódó belső összefüggéseket is hallja. Ezekben az utasításokban szándékosan, rendszeresen használom a programozásban szükséges fogalmakat: objektum, egész, szöveg, adat, algoritmus, tulajdonság, eljárás, függvény. További előny, hogy később, a hallott minta alapján, a diák meg tudja fogalmazni a kérdéseit.

Az ilyen ismertetések során a konkrét feladat általában nem értelmes. Grafikában absztrakt ábra készül, szövegszerkesztésben Lorem ipsum szöveggel, placebo objektumokkal dolgozunk. Nem egy példát kell megoldania, hanem kipróbál valamit legalább kétszer. Ezek a kipróbálások egyben az alkalmazás működésének tesztelését is jelentik, célzott kérdésekkel a tesztek elemzése a programozási ismereteket készíti elő.

Az alkalmazással ismerkedés során szoktam „Guess the Code” feladatokat adni. Ezek olyan feladatok, amelyekkel az alkalmazás működését vizsgáljuk, az alkalmazást készítő programozó döntéseit, az alapértelmezett beállításokat elemezzük; vagy éppen a digitális világ tulajdonságait tapasztaljuk meg. Néhány konkrét példát tartalmaz a XIV/1. melléklet. A feladatok tanórai alkalmazását bemutattam az ISSEP 2016-os konferencia workshopján [159], valamint a Színpadon a Természettudomány 2018-as fesztiválján [160].

Az eszköz kipróbálása után az első feladatok jellemzően a tanult szabályoknak megfelelő, de szabad felhasználását kérik. Ezt követi a leírás alapján történő feladatmegoldás, önálló egyéni, „mini-projekt” (XIV/2 melléklet) Mini-projekt formájában. A feladatok rendszerint tartalmaznak még nem tanult részeket is, azaz problémamegoldás is szükséges. Aki egy problémát meg tud oldani, az a megoldását – kérdésre – elmondhatja egy-egy társának. Akinek én segítek egy részfeladat megoldásában, azt néha fel is kérem, hogy adja tovább, amit mondtam. Aki kész van, az társai munkáját is figyelje, ellenőrizze, hogy minden részt jól oldott-e meg. A másoknak segítés szabálya, hogy csak szóban lehet segíteni, azaz nem veheti át az egeret, billentyűzetet. Szóban, a már kész megoldást nem(!) felhasználva kell kommunikálni a kérdést, problémát, a hibát, a javítási módot. Ez a módszer annak is tanulás, aki segít és annak is, akinek segít.

Nehezebb probléma felmerülése esetén, van, akinek útmutatót adok, van, akinek külön tanítom. Az élbolynak – 4–6 diák – én segítek, így ők tovább tudnak haladni, de később felkérem őket az elszórtan „befutók” segítésére. Van, hogy a problémát átteszem a következő óra elejére és akkor mindenkinek egyszerre tanítom – utána persze kisebb csoportokban újra megbeszéljük azokat a részeket, ami a frontális módszerrel nem ment át. A lemaradóknak is inkább én segítek, mert ezeknél a diákoknál speciális problémák is lehetnek.

A tanórák elején „soha nem tudom, hogy mi volt előző órán”. Az első feladat az addig megoldott feladatok átnézése. Különösen akkor, ha széthúzott a társaság, fontos, hogy mindenki ott folytassa, ahol abbahagyta a feladatot; ha otthon is dolgozott rajta, akkor ellenőrizze a megoldásokat, feltegye a kérdéseit.

A bemutatás LAU^{L}1, a kipróbálás a LAU^{L}2-3 fázisnak tekinthető, a kötetlen felhasználás LAU^{L}5a, a leírás alapján történő felhasználás jellemzően LAU^{L}5b szintű ismeretét jelenti az eredeti tananyagnak. Ezen belül, az egyes részletek megtanulása a részfeladatok mentén történik, a tanórák elején az áttekintés az adott ismeret LAU^{L}4 vagy LAU^{L}5 állapota. Ha folyamatosan lennének a tanórák, akkor is időszakosan meg kellene állni, és visszanézni a korábbi tevékenységeket. A mostanában jellemző heti egy tanóra – ami a különböző szünetek miatt esetleg csak havi egy alkalom – miatt a LAU^{L}4 felejtés fázisa dominánsabb, az áttekintő ismétlésre hosszabb idő, esetenként egész tanórák szükségesek. Ilyenkor újra meg kell csinálni a feladatot (nem csak átnézni), vagy egy másik, hasonló feladat szükséges.

A digitális írástudás kreatív felhasználásához versenyfeladatot vagy „éles” alkalmazást szoktam kiadni. Ez általában több órás feladat, amit otthon is lehet „fejleszteni”.

A házi feladat jellemzően LAU^{L}5a típusú vagy a hiányok pótlása. Kivétel az érettségi felkészülés időszaka, mert ott a feladatmegoldás a házi feladat, a tanórán csak értékelünk, kommunikálunk az alternatív megoldásokról, felmerülő kérdésekkel kapcsolatban.

A dolgozat LAU^{L}5a-5b szinten a tanórai feladathoz hasonló jellegű feladat. A LAU^{L}5c szinten tipikusan projekt jellegű feladatot várok el. Ha a feladat kötelező, akkor a határidő pillanatában lévő állapotot értékelem, de hetente 1 jegy „-kötbér” mellett lehet javítást beadni.

Tagozatos csoportban az alkalmazás önálló tanulását is tudtam tanítani. A csoport tagjai több, mint egy tucat alkalmazás közül választhattak – jellemzően grafika, 3D tervezés, filmkészítés vagy weblapkészítés témában. két–három hónapos időszakban a tanórákon és házi feladatként a választott alkalmazást – dokumentáció alapján – kellett megismerni és egy–öt projektmunkán bemutatni a használatát. Az alkalmazások önálló megismerése nagyon fontos lenne, de évek óta nincs rá idő.

8.1.3 Problémamegoldás informatikai eszközökkel témakör

A témakörhöz tartozik a táblázatkezelés, az adatbáziskezelés, a programozás oktatása. Mindegyiket először alkalmazásként tanítom, azaz a tanórai munka megszervezésében kezdetben nincs eltérés. Amiben más, az a megbeszélendő feladatok típusa és a megbeszélés módja. Míg a Digitális írástudás témakörhöz csoportosított ismereteknél a kérdés jellemzően az, hogy „hogyan csináljam", a problémamegoldás során a kérdés lényege a „hogyan értsem", „hogyan gondoljam". A diákok figyelmét fel kell hívni a váltásra. A megoldást elsősorban nem a menüben kell keresni, hanem ki kell találni, hogy mit keressünk a menüben.

Ebben a témakörben az a fajta feladatmegadás [75], hogy „számold ki SZUM függvénnyel az összeget", a LAU^{L}1-3 szintnek felel meg, nem problémamegoldás, hanem digitális írástudás. Ha a kérdés úgy szól, hogy „Hányan voltak összesen?", akkor már meg kell oldani egy értelmezési problémát, – ami sokak számára tényleg nehéz feladat, – hogy összeadni vagy megszámlálni kell az adatokat [131].

Informatikaórán informatikát tanítok, ezért mindig olyan feladatot adok, amibe az értelmezés is beletartozik [37, 32]. Nem adok konkrét eszköz használatát előíró feladatot, a specifikáció a diák feladata. A példa alapján, először egy feladat megoldásához szükség lesz az összeadás és a megszámlálás egyikére, a szükséges függvényt segítek megkeresni és megmutatom a használatát. Amikor egy másik feladatban a másik függvény kell, akkor észre kell venni, hogy az addig tanult eszköz nem megfelelő. Ekkor keresünk egy másik függvényt, a használatát összehasonlítjuk a már ismerttel. (Például így felfedezzük fel a DARAB2 és ÁTLAGA függvényeket is, szóba kerülnek az adattípusok – szám, szöveg, logikai.)

A problémamegoldást számonkérő első dolgozatban gondot szokott okozni, hogy a feladatot értelmezni is kell, mert nincs megadva, hogy melyik függvényt kell használni. A megoldás eszköze és ezek használatának módja is kérdés. A számonkért tudás nem magolható, ami a magoláshoz szokott diákoknak elsőre traumatikus. Ezért táblázatkezelésből az első 1-2 dolgozatból próbadolgozatot szoktam íratni, ami lényegét tekintve dolgozatkörnyezetben írt önálló gyakorlás. A következő, 2. vagy 3. dolgozat az értelmezést teszi előtérbe: egy versenyfeladatból vagy más, az aktuális ismereteket jóval meghaladó feladatsorból kell kiválogatni azokat a részfeladatokat, amelyekre tudja a választ. A pontozás a használt ismeretek mennyiségétől függ.

Az e-hód versenyen [141] minél több diákomat elindítom, de erre határozottan nem készítem fel őket, hogy a feladatok ne legyenek megszokottak, minél inkább kreativitást igényeljenek.

Hasonló megfontolásból ajánlom diákjaimnak a „barátságos” versenyeket és vetélkedőket, például a „<19”[28], az NJNT Programtermék[29] és a robotika versenyeket, valamint a Geek--Hertz [90] online versenyt, amit egy indiai iskola szervez a náluk oktatott témákban.

8.1.4 Programozás oktatása

A programozás oktatását azért veszem külön, mert itt válik dominánssá az informatika művelése az informatika alkalmazásához, a digitális írástudáshoz képest.

Az alsós robotika a korlátozott eszközkészlet (sablonok) miatt alkalmazásként tanítja a programozást. A blokknyelvek tanításakor az objektumok (szereplők) grafikai előállítása néha nagyobb része a feladatnak, mint a programozás, miközben a program javarészt szekvencia, amivel a program egyenrangúvá válik egy PowerPoint animációval. Emiatt a blokknyelvű programozás is sokszor digitális írástudásként érzékelhető. A robotikán belül a robotok építése egész más területre viszi el a feladatot, annak nagy része statika, elektronika, mechanika, anyagtechnológia. A problémák ezeken a területeken jelentkeznek, nem a programozás részen.

Amikor programozás útján oldunk meg problémákat, akkor a fejlesztőkörnyezet és a programozási nyelv (lehet blokk nyelv is) eszköz a megoldáshoz. A digitális írástudás része lehet, hogy egy fejlesztőkörnyezet használatában kiismerje magát az ember vagy, hogy egy programozási nyelvet értsen, de a témakörben nem ezeken van a hangsúly, hanem a programozási problémák megoldásán.

A digitális írástudás témakör oktatásába, ahol csak lehet, becsempészem a programozással kapcsolatos ismereteket, a programozás oktatásában ezt felhasználom, ezért a programozást LAU^{L}1 szinten – jellemzően – a programozás oktatása témakör előtt (esetleg évekkel korábban elkezdve) tanulják a diákok. A közoktatásban 2, 4, 6 esetleg 8 éven át építkezünk, a tanterv spirális felépítésén túl is bőven van lehetőség a programozás alapjainak előkészítésére.

A tananyag feldolgozását, ahol csak lehet, „mini-projekt”-ként (XIV/2) szervezem. Egy probléma megoldása – akár alkalmazói szoftver használatát igényli, amibe programozási ismereteket is vegyítek, akár programot kell készíteni, amihez a megfelelő fejlesztő környezetet használunk, – tekinthető projektnek, aminek a részei: igények felmérése, megvalósítás tervezése, kivitelezés, ellenőrzés.

A XIV/3. mellékletben az elnyújtott-átfedő LAU-okra mutatok példákat. Ezekben az ismertetés, magyarázat után évekkel várom el a tudást. Nem várom, hogy a diák megtanulja, ehelyett

[28] <19” Szabadfogású Számítógép Verseny: https://www.verseny.c3.hu/2021/#verseny [2021.10.30]
[29] Neumann János Nemzetközi Tehetségkutató Programtermék Verseny: https://njszt.hu/hu/page/neumann-janos-nemzetkozi-tehetsegkutato-programtermek-verseny-kovacs-gyozo-szellemeben [2021.10.30]

a többszöri előfordulás során – hasonlóan az informális tanuláshoz – rögzül az ismeret, a formális oktatásban a LAU^{L}1 fázis az lesz, amikor először kontextusba kerül a többi megtanulandó ismerettel. Jellemző, hogy ekkor a megértés Active lesz (LAU^{L}1A). A programozás oktatásában tipikusan ilyen az elemi adattípusok fogalmai, az objektum, tulajdonság, függvény, eljárás fogalmak.

A LAU^{L}2 szinthez, a kipróbáláshoz meg kell ismerkedni a fejlesztőkörnyezet megfelelő szolgáltatásaival – ahogy bármelyik alkalmazás esetén – és a nyelv szavaival – ahogy táblázatkezelésben a függvényekkel.

A programozás téma első órája az eszközök – fejlesztőkörnyezet, nyelv, fordítóprogram szerepéről szóló beszélgetés, nézelődés. A második órán az alkalmazásokhoz hasonlóan, vezetett gyakorlatként kezdjük a programozást. Az első programunk nem a „Hello World”, hanem a „Hogy vagy” szöveget írja ki. Nem szoktam kivetíteni. Itt sincs prezentációm, de mivel itt karakterre pontosan kellenek szavak, nem csak kimondom, hanem a táblára felírom. Így a legelső lefutó program is saját alkotás eredménye.

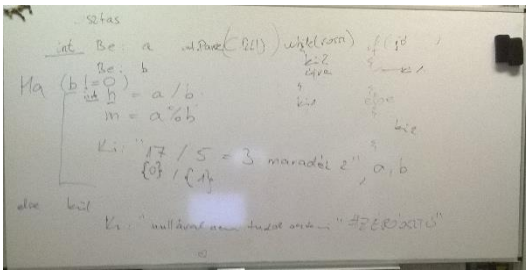
A program megállítását breakpoint beszúrásával tanítom – mert a jó programok a végén maguktól képesek bezáródni, továbbá azért, mert ezzel a program tesztelése felé is teszünk egy lépést.

Amint láthatóvá válik a „hogy vagy” kérdés, a diákok – lányok inkább – igénylik, hogy válaszolhassanak. Tanóra végére eljutunk odáig, hogy szövegbevitel, a szöveges adat tárolása, a szöveg kiírása és a választól függő feltétel szerepel a programban. Mindenki egyénileg fejlesztheti a felhasználóval kommunikáló programját.

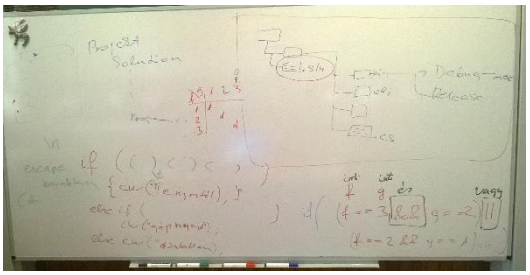
A következő órákra több feladatot viszek, amiből a csoport választ. Lehet két különböző választás is. Ennek megfelelően kerül sorra a háromféle ciklus valamelyike és a véletlenszámgenerátor használata. Közben az algebrai és logikai műveletek, az értékadások is szükségessé válnak. Többórás feladatot is kezdetben minden órán újrakezdünk, az első változat után továbbfejlesztjük. Lehet egyéni ötleteket is beépíteni.

Eközben sokszor a gépemet sem kapcsolom be, mindenki önállóan alkot. Az új nyelvi elemeket a táblára írom, sokszor nem is egybefüggően. A diák feladata megtalálni a kódban a helyét. Azaz megtanul tájékozódni a kódban, megtanulja fejleszteni a programját és a lépésenkénti tesztelést. Jellemző, hogy óra végére a programok különbözők lesznek, mert a fejlesztés különböző irányait választják a diákok. Szintén jellemző, hogy a táblakép csúnya, mert az igények indukálta bővítésből következően nem tervezett, ugyanakkor lehetőleg nem törlök le semmit a tanóra végéig.

Példák táblaképre:



9. ábra: Összeadás után önállóan az osztás C# nyelven



10. ábra: Első játékprogram: kő-papír-olló C# nyelven

Amikor a feladat többórás, már gyakrabban írom én is a kódot, főleg a hiányzók miatt, hogy pontosan meg tudjam mondani, mi volt órán. Ilyenkor óra alatt írom a megoldást, az alapján, ahogy a diákokkal megbeszéltem a részleteket. Csak az óra végén vetítem ki, ha szükséges. Óra közben oda lehet menni, a tanári gép monitorjáról meglesni, ellenőrizni a megoldást, de nem lehet jegyzetet, fényképet készíteni (és a betűméret olyan kicsi, hogy senki sem látja a helyéről). Ezzel érem el, hogy ha másolja a kódot, akkor azt meg is kelljen jegyeznie. Aki sokat járkál – inkább magol, mint kitalálja a megoldást –, annál ellenőrzöm, hogy érti-e. Amit nem ért, azt kitörlöm.

Az első dolgozat egy egyéni, otthon megtervezett feladat tanórai programkészítéssel. A minimum: egy adat bekérése és tárolása, egy vezérlési szerkezet használata, eredménykiírás. Mindvégig alapelvem, hogy nem adok mankót a gondolkodáshoz, mert az gyengít, lustává tesz, de ha hiányzik valamilyen ismeret, azt elmondom.

Leginkább probléma-orientált [150] módszert használok, de igyekszem minden irányba kitekinteni. A digitális írástudás tanítása alatt is néha az adatok, máskor az algoritmusok kerülnek előtérbe. Ahol volt rá idő, ott a rajz témakörben kitekintésként adtam keretprogramot és belenéztünk a .bmp kódjába, így lehet kódolással rajzolni, problémaorientált módszert követve. Az érettségire készülőknek biztosan adok néhány nyelvorientált feladatot. Volt olyan csoportom, ahol a tipegő szint után a statisztikai függvények megvalósításával folytattuk és végigvettük az algoritmusokat, a módszeres programozás alapjait. Másik csoportban a OOP alapozásához a „Baba” osztály követte az első programokat, amibe fájlból olvastuk be a játékbabák/újszülöttek – ki mit akart – adatait. Matematikatagozatosoknak a prímszámok keresése, majd ezt követően a matematikaórán tanult algoritmusok felé mentünk el. A konkrét felépítés és módszer alkalmazása helyett inkább az érdeklődés dönti el a választott módszert, de időszakosan, például összefoglalásként, új feladatok ajánlásakor váltok. Az érettségire készülés útján sorrakerülnek a programozási tételek és az osztályok is. A tanórákon figyelem az önállóság mértékét, ennek függvényében lehet egyéni érdeklődésnek megfelelő programot

is írni, esetleg grafikus programot írni és otthon folytatni. Az emelt szintű érettségi előtt programozásból is el kell érni, hogy 90 percen belül (ezért otthon), önállóan írjon programot a diák, a tanórán a kérdéseket és a különböző megoldási módokat beszéljük meg. A cél nem a profizmus, hanem hogy képes legyen önállóan képezni magát.

A programozás tanulása a programozás mikéntjének a felfedezéséről szól. A problémától kiindulva haladunk a megvalósítás felé, a problémamegoldás igényétől vezérelve, induktív szemlélettel. A cél, hogy elérjük az egyetemi absztrakció alap szintjét és a programozásról mint tevékenységről legyen a diáknak tapasztalata – ebből tud az egyetemen deduktívan szakmai ismeretekre és gyakorlatra szert tenni.

8.2 Mentorálás, motiválás[30]

Az önálló feladatmegoldást, informatikai gondolkodást nem lehet magyarázattal bemutatással megtanítani. Mivel ezek belső (az agyban végbemenő) tevékenységek, nem lehet közvetlen megfigyeléssel tanulni. Amikor egy gondolatmenetet elmondok a diáknak, akkor nem azt figyeli, hogy hogyan gondolkodom, hanem azt próbálja megjegyezni, hogy mit mondok.

Amikor egy diák megakad és kérdez, azt kell figyelni, hogy a kérdés mögött milyen probléma húzódik meg. A megoldáshoz szükséges ismeretek milyen mélységű tudása mellett teszi fel a diák a kérdést. Ehhez legjobb visszakérdezni, elmondatni a diákkal, hogy eddig mivel, hogyan próbálkozott. Sokszor ennyi elég is, mert közben maga is rájön a megoldásra. A kimondás, a hangzó megfogalmazás rendezi a kavargó érveket. Ha nem tudja elmondani, hogy mi okoz félreértést, akkor érdemes leíratni, rajzoltatni, ez még inkább segíti a gondolatok, érvek rendszerezését. Ilyenkor a háttérben arról van szó, hogy a diák számára a feladat túl összetett ahhoz, hogy fejben megoldja, vagy a probléma forrásait meghatározza. Az elbeszélés, a leírás a specifikálást, a tervezést pótolja. A tanár szerepe ilyenkor a meghallgatás, a kontroll és szükség esetén a hiba jelzése, illetve a bizonytalan vagy hiányzó pontokon a megerősítés, kiegészítés.

A tehetséges diákok ezt a mentorálást igénylik leginkább. A nagyon tehetséges diáknak is fontos, hogy elmondja, mit gondol, mert ezzel ő, önmagában tisztázza a kapcsolatokat. A tanár dolga, hogy megértse, vagy ha nem érti, akkor visszakérdezzen, részletesebb igazolását kérje a „zseniális meglátásnak”. A tanár önképe szempontjából fontos tudni, hogy attól, hogy megérti, mit mond a diákja, ő nem fogja tudni, – egyszeri hallásra nem fogja megtanulni, – a magyarázat

[30] T1, T2 tézisekhez: Az informatikai gondolkodás tanításának a módszereiről szól ez a fejezet. Az informatikai gondolkodás tanítása az informatika tudomány oktatásának legfontosabb kompetenciája. A T4 tézis alapja, annak tisztázása, hogy mi segíti, gátolja az informatikai gondolkodást.

lényege az, hogy a diák megtanulja azt, amit kitalált. Ebből a szempontból egy tanár és egy diák magyarázata egyforma hatású.

Azt tapasztaltam, hogy a módszer a legkevesebb adottsággal rendelkező diák számára is hasznos. Azzal, hogy a diák elmondja, önmaga számára felfedezi, rögzíti az összefüggéseket, fogalmakat. Közben megfogalmaz kérdést a hiányzó részletre.

Amennyiben a kérdezett részlet a feladat szövegében megtalálható, akkor azt az információt kell megadni: „olvasd el még egyszer a feladat elejét” (vagy a kérdést, a tábladefiníciót); „nézd meg a mintát”. Figyelni kell arra, hogy ne a konkrét választ adjuk meg, mert a probléma az adatok kigyűjtésével, a feladat értelmezésével van.

Ha a tanultakkal kapcsolatos a hiba, akkor sem az az első, hogy elmagyarázzam (újra) a szükséges ismereteket. Menüelem vagy beállítás keresése esetén az a visszakérdés, hogy logikusan hol kellene lennie, hogyan lehet megtudni a választ, milyen segítséget ad a szoftver. Ha a probléma az, hogy nem működik, nem úgy működik a szoftver, ahogy szeretnénk, akkor a működést befolyásoló tulajdonságokról kell kérdezni. Szövegszerkesztés esetén – például – nem szabad a kérdést sem meghallgatni, amíg nincs bekapcsolva a diák munkájához a „mutat/rejt”, a nem-nyomtatódó karakterek nem láthatók.

Szintaktikai hiba esetén – kódolás, weblapkészítés, táblázatkezelés – nem a hiba kijavítása segít, hanem az, ha megkérdezzük, mit mond a „gép”. Hol jelzi a hibát, milyen hibaüzenetet ad – ebből mire lehet következtetni.

Ha a tanultakkal kapcsolatos hiba a tanultak megértésének hiányára utal, akkor rá kell kérdezni arra, hogy az adott témából mit tud, mire emlékszik. Nem szabad elkezdeni elölről levezetni – vezesse le a diák – és nem szabad minden részletet kifejteni. A diák tudásához illeszkedő rövid javítás, kiegészítés az, ami segít, utána egy nagyon rövid vázlat kíséretében a diák egyéni tennivalóihoz (todo listájához) kell sorolni a további részletek áttekintését. Ez nem feltétlenül házi feladatot jelent, mert a feladatok folytatása során is sorra kerülhet a teljesítés, viszont témától függően javasolni lehet feladatokat is mellé, ami nem kötelező.

A direkt problémajelzések egy része motivációs problémát jelez: „én ezt úgy sem tudom”, „tessék megmondani, hogy mit csináljak”; de gyakoribb, hogy indirekt jelzésként jelentkeznek a gondok. A diák „rendetlenkedik”, játszik, piszkálja társát, beszélget a társával, csak néz maga elé. A jelzések egy része arról szól, hogy a diák figyelmét elvonja valami, másik része az önértékelésével kapcsolatos, de így jelentkeznek különböző másságok és a fizikai akadályok is.

A diák „figyelmét elvonja valami” állapotnak nagyon komoly háttere is lehet. Egy családi tragédia, csalódás a barátban, fontos bizonytalan hatású eseményre vár (például nyelvvizsga-eredmény) vagy játékfüggő és a játék folytatásán jár az agya, vagy egy kommunikáció érinti érzékenyen. Ezeket a munkát akadályozó eseményeket célszerű felderíteni és helyzettől függően segíteni abban, hogy félre tudja tenni vagy támogatni a nyugvópontra jutásban [40].

A diák figyelmét maga a feladat is elvonhatja. Például kódolás közben az algoritmusra kellene figyelnie, a kódolandó tartalom a lényeg. Ha a diák nem tud jól gépelni vagy a billentyűzeten nem találja a ’[’ karaktert, akkor a figyelmét meg kell osztania az algoritmus és a bevitel között. Ez nehezíti, lassítja a munkát, idegesítő, frusztráló, „ez így nem megy” eredményt hoz. Ilyen esetben az a fontos, hogy tisztázzuk, mi az, ami a munkát gátolja, a zavaró képességhiányt kell először megszüntetni – a példa esetén a gépelést külön gyakoroltatni.

Az önértékelési gondokat meg kell beszélni. A maximalista azért nem tud dolgozni, mert azonnal akarja látni az eredményt. Nagyon gyakori, hogy ezek a diákok a magolásban bíznak: eddig is mindent meg tudott tanulni, a feladatok megoldását is meg tudja tanulni. Számukra frusztrációt jelenthet, ha valamit ki kell találniuk, mert az azt jelenti, hogy nem tanulta meg.

Azok, akik gyengébbnek érzik magukat a társaiknál, szeretnének más területen kitűnni: a legviccesebb, a legügyesebb hacker. Sokszor csak a figyelem igénye – szeretethiány – jelenik meg akár a tanár felé, akár a társa felé.

Tipikus eset, hogy a diák fél dönteni, minden részfeladat után megkérdezi, hogy jó-e. A válasz: „Szerinted?” Ha több lehetőség között nem látja, hogy melyik a jó (jobb) megoldás, akkor ki kell próbáltatni mindegyik felmerülő ötletet. Fontos, hogy a megerősítés tapasztalatszerzéssel történjen és ne tanári visszaigazolással, így megtanulja saját ötleteit értékelni, vállalni a döntéseit.

Fizikai akadály lehet például egy fárasztóan fényes monitor, a bevilágító napfény, a rossz egér – ami miatt nem az történik a programban, amit a diák gondol –, a meleg, az oxigénhiány… De fizikai akadály lehet egy kézsérülés vagy a szem romlása is.

Látszólag nem tartozik az informatika oktatásának módszertanához a „hozott állapot”, azonban az informatikai gondolkodáshoz az adott témára kell fókuszálni A „nem vagyok képes programozni” sokszor csak a fókuszálás képtelenségét jelenti. Az egyik legnehezebb dolog nem gondolni valamire, „félretenni” az érzéseket, gondolatokat.

> *„A profi programozók úgy osztják be a szabadidejüket, hogy a munkahelyen töltött idő a lehető leghatékonyabb legyen. Ez azt jelenti, hogy otthon arra is kifejezetten*

időt kell szakítanod, hogy kiküszöböld a feszültséget okozó tényezőket, hogy a magánéleti gondjaidat ne vidd magaddal a munkahelyedre.” [157 p. 79]

„A programozás szellemi gyakorlat, amely hosszú távú összpontosítást igényel. A fókusz azonban ritka kincs – olyan, mint a manna. Ha elhasználtad a "fókuszmannádat", legalább egy órán át összpontosítást nem igénylő tevékenységeket kell végezned, hogy újratöltsd a raktárakat.” [157 p. 140]

Egy diák nem feltétlenül rendelkezik a szabadidejével, nem tudja tanórán kívülre helyezni a feszültségeket. Főleg nem azokat a feszültségeket, amelyeket pont az informatikaóra, a tanulással kapcsolatos elvárások okoznak. Informatikát csak úgy lehet tanulni, hogy a feladatra koncentrál a diák, ezért az informatikaoktatásnak része, hogy a koncentrálást segítse, a koncentrálás gátló tényezőitől segítse megszabadulni a diákot. Az informatika és a programozás tanulását segíti, ha a feladat témája érdekes vagy aktuális. Az önálló gondolkodás, a feladat megoldása – mint egy rejtvényé – önmagában motivációs tényező, de a siker kulcsa sokszor a megoldások megtalálását gátló tényezők hatástalanítása.

Amennyiben a diák a megoldás kitalálása helyett tanárától vagy társától megkapja a megoldást, az rövidtávon gyorsan elkészített feladatot jelent, de hosszú távon önálló megoldásra képtelenséget eredményez.

A másságokra külön figyelni kell, bármilyen típusú legyen, a diákkal meg kell beszélni, hogy ő hogyan kezelje a feladatokat, kialakítani az egyéni tanulási stratégiájukat. Például nem szabad azt néznie, hogy a társak valamiben gyorsabbak. Van, akinek külön időt kell adni, hogy a tanórai anyagot befejezze, van, akinek a végén ellenőrizni kell, hogy pontosan megcsinálta-e a feladatokat, nem hagyott-e ki néhány pontot, van, akinél a feladat elején kell odafigyelni, hogy jól értette-e a feladatot.

Az általánosan elvárthoz képest másképp tanulók (XIV/4) számára a nekik megfelelő formában kell felépíteni a tananyagot. A legismertebb specialitások:

- A programozási nyelvek tanulása a diszlexiásoknak a nyelvtanuláshoz hasonlóan nehéz, de megfelelő tananyagszervezés és a munkát kódkiegészítéssel támogató eszköz választása sokat segít.
- A diszgráfia kompenzálható helyesírást támogató (szókiegészítő, hibajelölő, automatikus javító) eszközökkel lehetséges.
- Mivel az informatika nem matematika, a diszkalkuliás informatikus is számos területen tud alkotni, megfelelő feladattípusok szükségesek az informatikai képzéshez.

Az Asperger-szindróma (és más autizmus spektrumzavar) nem csak az emberekkel, hanem az eszközökkel való kommunikációban is speciális megnyilvánulásokban jelenik meg. Kiemelkedő teljesítmény, egészséget is veszélyeztető függőség, de dühroham és depresszió is kísérheti az informatika tanulását.

8.2.1 Önértékelés szerepe

Az informatikus szakmában nagyon egyenlőtlen a nemek aránya. A Microsoft felmérése [158] azt mutatja, hogy a lányok 15-16 évesen veszítik el az érdeklődésüket a STEM iránt. Az informatikatanítása szempontjából lényeges elem, hogy a kamaszkorban előtérbe kerülő nemi szerepek és társadalmi elvárások is közrejátszanak a lányok gondolkodásának fejlődésében. Ezt a jelenséget elemeztem a XIV/5. mellékletben. A vizsgált esetek azt mutatják, hogy a lányokra a megvizsgál, mérlegel, megjegyez, tervez tevékenységek (ebben a sorrendben) jellemzőbbek, míg a fiúkra a szétszed, megvizsgál, kitapasztal tevékenységek jellemzőbbek. A lányok a biztonságot hátra húzódva keresik, a fiúk kiharcolják maguknak, megvédik a területüket. Informatikai feladat megoldása során, amikor egy fiú és egy lány együtt dolgozott, ötből ötször azt tapasztaltam, hogy a fiú kommunikál a géppel, a lány kontrollál, kérdez, óvatosan tanácsot ad… nem a géppel kommunikál, hanem a fiúval. Ez független attól, hogy kinek a keze írja a kódot.

Páros munka során – nemektől függetlenül – gyakori, hogy az egyik lesz az, aki kigondolja, a másik a háttérmunkás, kontrollálja a megoldást, vagy „csendes társ”. Idővel az, aki kigondolja azt fogja mondani magáról, hogy tud programozni, tudja az informatikát – mert bizonyította már számtalan esetben –, a háttérmunkás pedig úgy fogja érezni (vagy eleve azzal igazolja passzivitását), hogy nem tud programozni, mert mindig segítséggel készült el a megoldása. A tudással kapcsolatos önértékelés és a feladatmegoldási stratégia erősíti egymást. A fiús képességtudat a feladat megoldásának (átgondolás nélküli) elkezdésével erősítik egymást. A lányok önbizalomhiánya azzal, hogy a feladatot akkor kezdik végrehajtani, ha látják a célig vezető utat, szintén erősítik egymást.

8.2.2 Sikerorientált oktatás

Ahhoz, hogy a lányok számára az informatikus pálya vonzó legyen, sikerélményt kell biztosítani. Ahogy a programtervezőnek tudnia kell kódolnia, a mérnökinformatikusnak tudnia kell algoritmust terveznie, a lányoknak és fiúknak önállóan kell tudnia az informatikát, a programozást.

Ahhoz, hogy valaki jó csapatjátékos legyen, az alapokat minden részletében ismernie kell, képesnek kell lennie a teljes folyamat végrehajtására. Nem azért kell munkamegosztás, mert az

egyik fél csak az egyik területet ismeri, hanem azért, mert így hatékonyabb a munka. Bár ma az oktatásban a csoportmunka, a pármunka, az együttműködés a modern módszertan jellemzője, ezek bármelyikét csak az egyénileg képzett, egyenrangú felekkel szabad alkalmazni.

Azt tapasztaltam, hogy az informatikaoktatásban, főleg a programozás oktatásában ezért elsősorban az egyéni munka, az egyéni projekt az első. Csak a tudás önálló, saját részre való elsajátítása, a képességek megszerzése és meglétének saját részre történő igazolása után szabad közös munkában részt venni. Az informatikaoktatásban a csapatmunka akkor fejlesztő hatású, ha a csapattagok egyénileg elvégzik – ugyanazt – a feladatot, egyénileg gondolják ki a megoldást, majd a megoldásokat összehasonlítják, megvitatják.

Az oktatás során figyelni kell arra, hogy a tanuló összeszedje a bátorságát és elhagyja a biztos ismeretek alkalmazásának komfortzónáját – kezdjen bele olyan feladat megoldásába, aminek nem látja végig a kivitelezés útját. Kísérletezzen saját képességeivel. Ne használjon mankót, biztosítókötelet, segítőket, hanem vállalja a bukás kockázatát, tanuljon meg elbukni és felállni, a bukás tapasztalatát felhasználni. Ez adja az „én egyedül képes vagyok rá” sikerélményét. A sikerélmény kulcsa a feladatok megoldási módjának önálló kitalálása, az önálló kivitelezés és tesztelés – önálló megbizonyosodás arról, hogy a megoldás jó.

A lányok az előre biztosítással, a megoldási mód kitalálása helyett kérdezéssel az elsőre biztosan jó megoldás elkészítésére törekednek, általában jobban félnek a bukástól, mint a fiúk. A félelmet kell legyőzni, nem a bukást elkerülni. Elsősorban önmaguk számára kell bizonyítani, hogy egyedül meg tudják oldani a feladatokat.

8.3 Programozás tanulhatósága[31]

Az információs társadalomban a programozás képessége a társadalomban való aktív részvétel feltétele [4]. Számos fejlett ország köznevelési programjában szerepel, hogy mindenkinek tanulnia kell programozni, a programozási/algoritmizálási készség a digitális írástudás része, a jövő polgára számára nélkülözhetetlen. Ugyanakkor erősen hiszik egyes diákok, szülők, hogy nem mindenki képes megtanulni programozni. Láthatjuk, hogy a disz-esek az oktatási rendszer termékei, egy adott oktatási módszer adott típusú disz-es diák számára létidegen.

A kérdés, hogy van-e olyan típusú diák – és ennek mik a jellemzői –, aki egészségügyi problémával nem küzd, de semmilyen módszerrel nem képes az elvárt informatikai és programozási ismeretek, készségek minimumszintjét elsajátítani. (Ebből következően: fel kellene menteni a követelmények teljesítése alól.)

[31] T4 tézis részletes kifejtése

Az informatika az informatikai gondolkodás tudománya. Van-e olyan ember, aki képtelen informatikai gondolkodásra?

Minden középiskolai csoportomban volt/van olyan diák, aki azt állította magáról, hogy képtelen megtanulni programozni. Ezt általában a szülő is megerősítette, általában azzal kiegészítve, hogy ő maga sem tud, így nem tud segíteni a gyereknek. Egyes esetekben érvként hozzák, hogy a gyerek humán beállítottságú.

Szinte minden esetben a diák azt gondolta, hogy csak úgy tud tanulni, ha adok neki valami megtanulni valót és azt számonkérem. Azaz, a tanulás számára a magolást jelenti. Sokan szívesen vállaltak volna kiselőadást, otthoni gyűjtőmunkát, de ez jellemzően internetről összevágott szöveget jelentett volna (megkérdeztem, így gondolták), ezzel bizonyítva a szorgalmat. Néhányan szívesen dolgoztak volna együtt a többiekkel, mondván, hogy majd nézik, mit csinál a társuk, és abból tanulnak. Ennek az együttműködésnek egy tipikus esete, amikor órán folyamatosan segít a mellette ülő diák vagy barát, majd a dolgozatnál is így szeretne teljesíteni. Minden esetben a diák kimondta magáról, hogy ő „hülye ehhez”; sajnos minden esetben volt valaki, aki ezt megerősítette.

A programozás (de igaz az informatikára általában is…) tanítását minden esetben ezzel a legutóbbival kezdtem: szerintem nem hülye, csak kényelmesebb ezt mutatnia magáról. Ehhez olyan példákat mondok, amit az adott diák önállóan kitalált és azt mutatja, hogy értelmes.

A második lépés a társakról történő leválasztás. Ne utánozzon, ne kérjen segítséget, önállóan teljesítse a feladatot. Így jellemző, hogy a leggyengébb tanulók külön ülnek, és külön, egyéni feladaton dolgoznak. Nem az a lényeg, hogy a feladat készen legyen, hanem az, hogy teljes egészében a diák találja ki, hogyan oldja meg.

A harmadik lépés a magolás kizárása. A feladat megoldásához szükséges tudáselemeket megadom, de a megoldás módjára önállóan kell rájönnie. Jellemző, hogy a kérdésekre visszakérdezek. Például: „Ez jó?” – „Szerinted?” „Fut?” „A Debuggert már nézted?”. A hisztit leiratom. Például: „Nem tudom, hogy csináljam” – „Írd le, hogy mit nem tudsz, hogy hogyan csináld!” „Írd le részletesebben!”

Néhány eset:

- Három lány 10. évfolyamosként kijelentette, hogy soha nem fog kelleni nekik az informatika, végül mindhárman informatikából érettségiztek. A kettő között volt egy nagyobb vitánk arról, hogy a jövőben a szellemi vagy a testi adottságok lesznek-e jövedelmezők, majd arról, hogy szeretem-e őket, amikor nem engedem, hogy órán

másolják a megoldásokat, nem fogadom el minimumként mások gondolatainak a megtanulását.

- Egy fiút fél éven keresztül minden órán el kellett ültetnem az aktuális segítője mellől. Először a legjobbak között ült, azután a fiú osztálytárs mellett, majd a lány osztálytárs mellett – mindegyik segített neki, mindig azt írta, amit a szomszédja. Amikor egyedül maradt, 45 percen át semmit sem írt. Azután a feladat részeit megjegyzésben írta meg, majd eljutott odáig, hogy létre kellene hozni egy változót. Nosza! Óra végére írt egy ötsoros programot. A továbbiakban érettségi feladatrészleteket oldott meg. Mindvégig problémát jelentett, hogy nagyon nehezen kezdett el írni, minden egyes lépésen nagyon sokat gondolkodott – jó lesz-e, így vagy másképp kellene… Számítógépes grafika irányban tanul tovább.
- Egy fiúval több, mint egy évig küzdöttem, hogy ne más munkájával ékeskedjen… Amikor mindenkinek más feladatot adtam, kénytelen volt írni valamit. *n* magasságú *V* alakot kellett a konzolon megjelenítenie. Az első megoldása („\/”) után bíztattam, hogy írja meg 2 magasságúra is. Azután másolta a kódot, majd rájött, hogy ciklus kell. Otthon folytatta… azzal jött vissza, hogy sikerült és ez nagyon jó. Sikerült neki egyedül. Azóta gondolkodik azon, hogy programozó lehetne – mert azzal sok pénzt lehet keresni, és képes lenne rá. Sokszor szeretne magolással, kódrészlet másolásával előbbre jutni, de már ő is látja, hogy az az érték, ha sikerül megoldani egy problémát, ha el tudja másnak is mondani, hogy mit talált ki, ha a kész, általa tervezett és írt programjával játszanak a diáktársai. (Folytatás: felvették az ELTE IK PTI-re.)
- Egy hetedikes lány, aki mindent a barátnőjével közösen szeret megcsinálni, programot is közösen tudott csak írni – vagy úgy sem. Az első önálló programjában kiírta a konzolra, hogy nem tud programozni. Kitalálta, hogy előérettségizik informatikából, mert abban nincs programozás és akkor soha nem kell programozást tanulnia. Algoritmikusan végiggondoltuk ezt a folyamatot, így rájött, hogy az elképzelése rossz – például nagyon valószínű, hogy nyolcadikosként már az osztályozó vizsgákon megbukna, ráadásul abban benne van a programozás is. Algoritmizálós feladatokban ő volt a csoportban a leggyengébben teljesítő, de a hibáit megnézve, elemezve ő is látta, hogy apró figyelmetlenségeken veszített sokat. Problémamegoldást igénylő feladatokat csak akkor oldja meg, ha érdekli vagy bukás környékére kerül. Akkor viszont egész ügyes. A gyenge eredmény miatt a szülőkkel is beszéltem. Ők is rájöttek, hogy gyermekük nem „hülye”, már nem szeretnék, hogy felmentsem a gyereket a teljesítés alól.

8.4 A módszerek komplex alkalmazásának az eredménye

A 2. táblázat 6,5 osztály, 218 diák (a 2017c fél osztály, 31-35 fő/ osztály) egyes témakörökből elért dolgozatainak az eredményeit tartalmazza. Minden csoportot én tanítottam, a felhasznált értékelést én végeztem. Kivétel közöttük a 2012b_term6, amelynek többször volt tanárváltása: 7. és 9. évfolyamon én voltam a tanáruk, 8. és 10. évfolyamon egyik, majd 11. és 12. évfolyamon másik kolléga tanította őket, így jegyeik hátteréről kevés ismeretem van, de biztos, hogy programozást nem tanultak.

2. táblázat: Néhány osztály témazáróinak eredménye

	elm	**rajz**	**szov**	**prez**	**tk ala p**	**tk köz**	**tk ab**	**web**	**komm**	**doc+ stílus**	**ab sql**	**prog**
2011d_ált4	4,49	4,72	4,22	4,54	3,47	4,13	3,87					3,71
2011a_mat6	4,44	4,62	4,65	4,15	4,44	4,10	3,89	4,71		4,14	3,73	3,52
2012b_term6	4,29	4,61	4,38	?	?	?	?	?	?	-	-	-
2015a_mat6	4,62		4,49	4,42		4,05		4,72	4,22	3,98		4,72
2016b_term6	4,34	4,61	4,59	4,42	4,09	4,39	4,24			3,72		4,27
2017b_term6	4,22	3,97	4,08	4,54	4,54	4,47		4,52		4,40		4,36
2017c_ált4	4,39	4,31	4,06	4,36	3,92	3,46		4,10	4,21	3,63	4,22	4,41

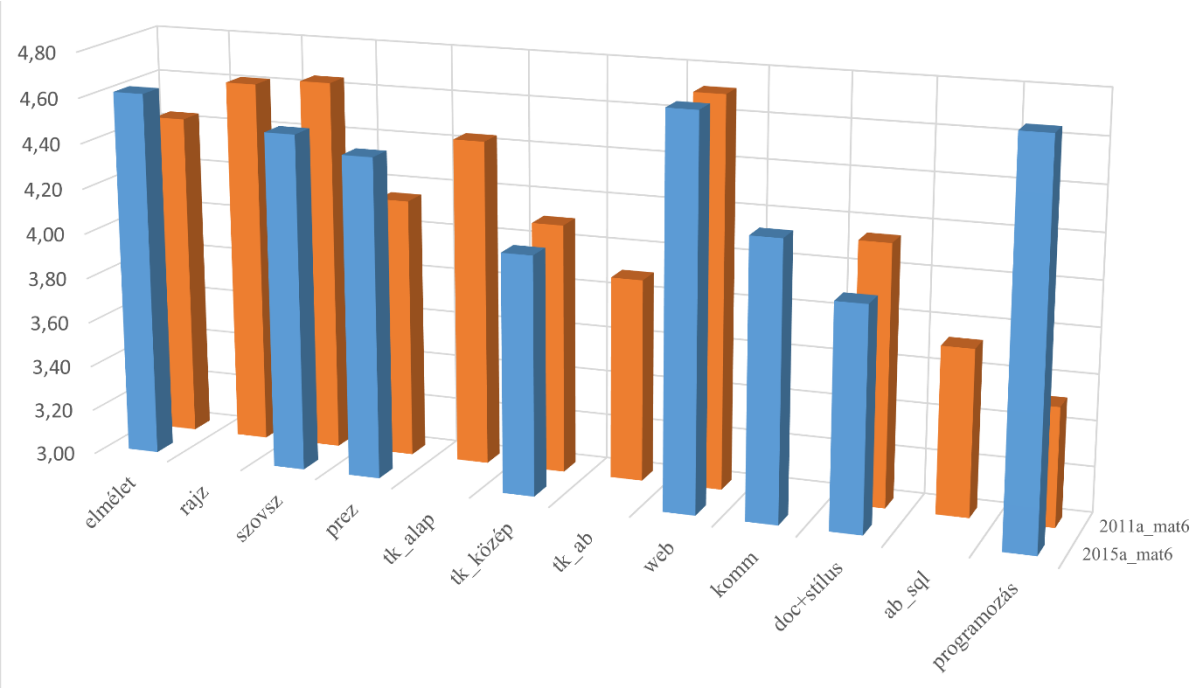
A XIV/6. mellékletben részletesen megtalálható az eredmények háttere, egyes csoportok programozás eredményének az elemzése és az egy évvel későbbi projekt feladat részeredményei.

A programozás dolgozat eredményén nagyon jól látszik a módszertani váltás:

- az algoritmusok, matematikai alapozás helyett a problémamegoldásra helyezett fókusz;
- az alkalmazások tanításába egyre tudatosabban beépített informatikai gondolkodás és ennek egyre tudatosabb számonkérése – akár a jegyek romlásának kockáztatásával is, ezzel összefüggésben a diákoktól a gondolkodás határozott, kérlelhetetlen elvárása;
- a programozás oktatása során a nem gondolkodók elkülönítése és gondolkodásra „kényszerítése”. (A 8.3 fejezet egyéni példái ezekben az osztályokban tanultak)

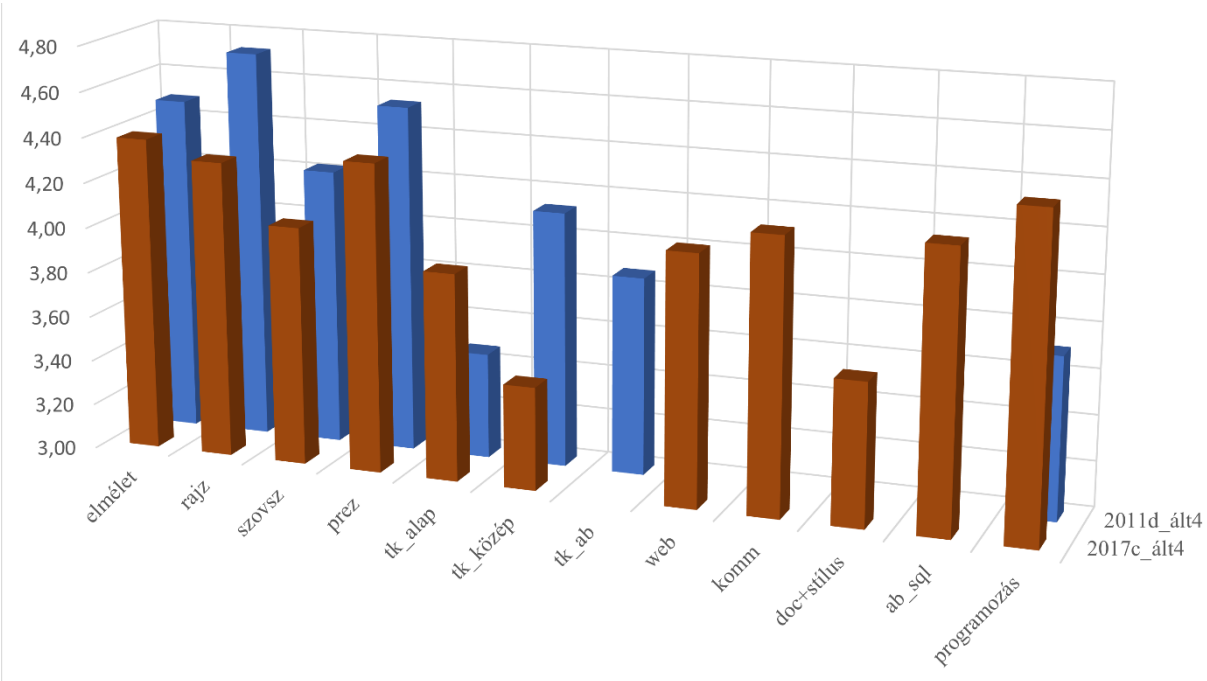
Képzési típusonként tekintve az adatokat, 2-2 osztály azonos felvételi rendszerben, azonos iskolában (szociális környezetben) tanult, a dolgozatok típusa is azonos volt. Szignifikáns különbség volt azonban, hogy a kutatásommal előtt vagy alatt tanítottam őket.

A két matematika tagozatos osztály dolgozatai átlagában, – ahol mindkét osztály megírta, – a programozást kivéve 4% az eltérés, a programozásban 30%-kal jobb, a későbbi osztály (11. ábra).



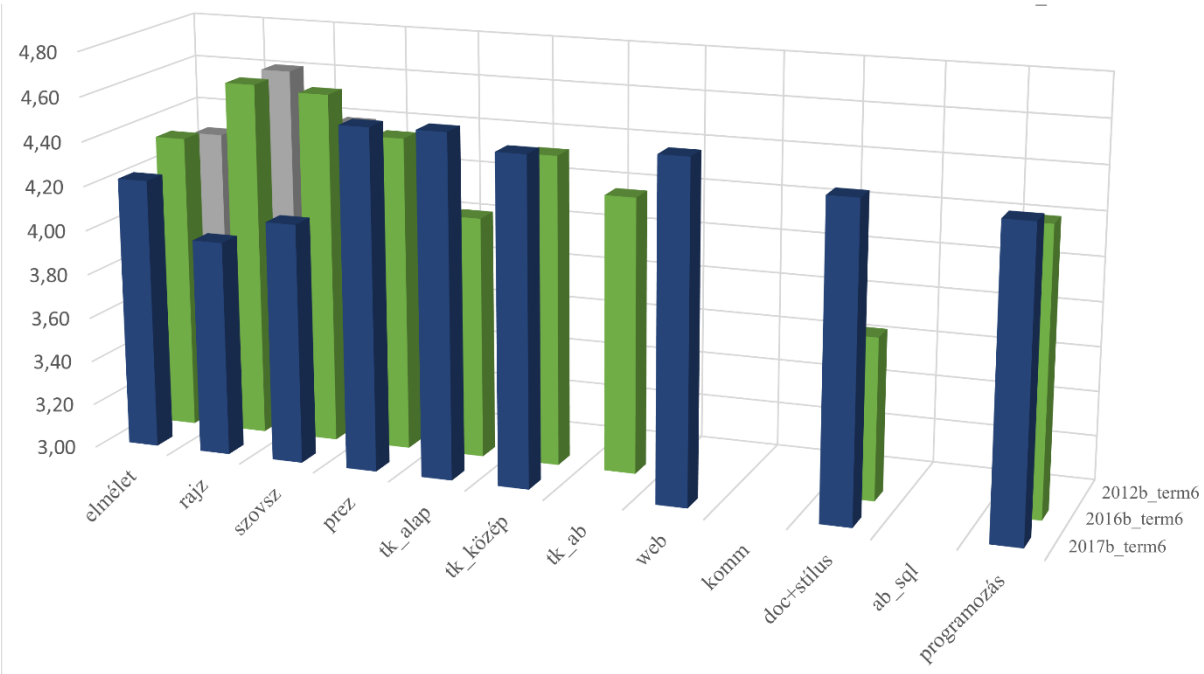
11. ábra: Matematika tagozatos (a) osztályok dolgozateredményei 2011-2014 és 2015-2016

A négyévfolyamos általános képzésű osztályok dolgozatainak eredménye, – ahol mindkét osztály megírta, – a programozást kivéve átlagosan 8%-ban tér el egymástól, a programozás a későbbi osztálynak 17%-kal sikerült jobban (12. ábra).



12. ábra: Négyévfolyamos általános képzésű (c, d) osztályok dolgozateredményei 2011-2013 és 2017-2020

A három természettudomány tagozatos (b) osztály dolgozatainak átlagában, – ahol mindhárom osztály megírta a dolgozatot, – az eredményben az eltérés átlaga 11%. A legfiatalabbak rajz és szövegszerkesztés eredménye (7. évfolyam) gyenge (13. ábra). A programozás nem értékelhető, mert a 2012-es osztály nem tanult programozást. A másik két osztály programozás eredménye közel azonos, 1-2%-kal gyengébb, mint az azonos korosztályú c osztályé.

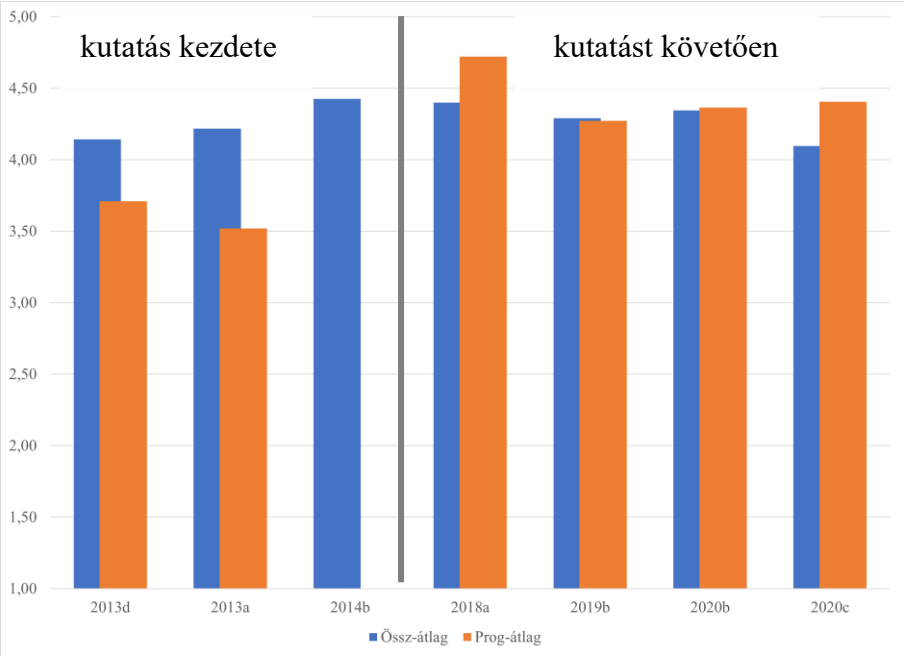


13. ábra: Természettudományos (b) osztályok dolgozatainak eredményei 2012-2015, 2016-2019 és 2017-2020

A programozás tanulásának sikerességét a többi témához viszonyítva látható, hogy a módosítások bevezetésével épp olyan megtanulható, mint a tantárgy többi témaköre. A korábbi, több, mint 10%-kal gyengébb teljesítményhez képest, az eredmény a többi eredmény átlagának megfelelő vagy jobb (3. táblázat, 14. ábra)

3. táblázat: Az osztályok programozás dolgozatának átlagai az összes dolgozat átlagaihoz viszonyítva

	Programozás éve	Összes dolgozat átlaga	Programozás dolgozat átlaga	Eltérés / Max. eltérés
2011d_ált4	2013d	4,14	3,71	-11%
2011a_mat6	2013a	4,22	3,52	-17%
2012b_term6	2014b	4,43		
2015a_mat6	2018a	4,40	4,72	8%
2016b_term6	2019b	4,29	4,27	0%
2017b_term6	2020b	4,35	4,36	0%
2017c_ált4	2020c	4,10	4,41	8%



14. ábra: Osztályonként – időrendben – a dolgozatok átlaga és a programozás dolgozatok átlaga

A programozás tanulás elemzésére a távoktatás a megszokottnál jobb feltételeket adott. A diákok nehezebben kommunikáltak egymással vagy külső tanácsadókkal. Amikor mégis, az azonnal észrevehető volt a beadott munkáikon. A 2016b és 2017c csoportok programozásdolgozatainak részletes elemzése megtalálható a XIV/6 mellékletben. Az iskola diákjainak az eredménye a kompetenciamérések során stabilnak mondható, a gimnáziumok rangsorában az első 10 helyezett között van. Ezért azt gondolom, hogy ez az 51 diák nem különbözik szignifikánsan attól a 35 diáktól, akiknek sikerült meggyőznie a kollégáimat arról, hogy nem képesek gondolkodni, nem tudnak megtanulni programozni. Az 51 diák között is volt, aki megírta, hogy „*Elnézést tanárnő, hogy nem csináltam meg a feladatot. Mondhatnám, hogy nem volt időm, de inkább azért halogatom, mert félek, hogy nem megy.*” 21 diák (40%) fejezte ki határozott félelmét a sikertelenséggel kapcsolatban és további 6 diák viselkedésével, ellenállásával bomlasztotta a csoportot, megpróbálták elkerülni a feladatok megoldását.

A 2016b osztálynak 2021-ben is volt informatikaórája. Tanév végén kilenchetes projekt feladatot kaptak: egyénileg választott témához esszédolgozat, prezentáció, weblap, táblázat, adatbázis és program készítése. A témákra részjegyeket kaptak, de végső jegybe a hat témából csak az öt legjobb számított be. A projekt leírása, a feladat megoldásának körülményei és az osztályra számított statisztikai eredmények a XIV/6 mellékletben olvashatók.

Két évvel a projekt előtt tanultak programozni, akkor a jegyek átlaga 4,27 volt. Ez mostanra 3,97-re kopott. Táblázatkezelés 4,39-ről (ennek felelt meg a projektben elvárt szint), 4,03-ra.

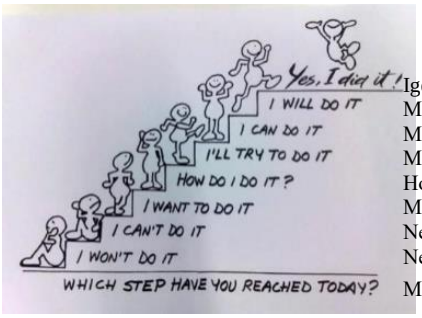
Egy diák hagyta ki a program megírását. A programozást néhányan szeretik, mások nem szeretik, de ha kell, mindenki tud egyszerű („tipegő” szintű) programot írni.

A vizsgált csoportokban a tanítási órák száma a kutatás elején a NAT-ban megadotthoz képest kevesebb, a programozásra fordított óraszám körülbelül fele a kerettantervben adottnak és nem épülhetett az általánosiskolai robotika ismeretekre. Ennek ellenére a programozás ismeretek a NAT elvárásainak megfelelnek, továbblépésre alkalmasak.

Az informatikai gondolkodásra készséget a diákokra egyénenként vizsgáltam. Kreativitást igénylő probléma felvetéséhez ismerni kell a diák „előéletét”, a gondolkodási tevékenység minősítéséhez emellett a megoldási motiváció, a tevékenységet befolyásoló lelkiállapot és a megoldási mód ismerete is szükséges. A 218 diákból a 2015-2017 között kezdő osztályok 117 diákjánál egyenként figyeltem meg a problémamegoldás informatikai alkalmazását. 14 diáknak nem volt „természetes” ez a gondolkodási mód, nekik „drukkoltam” és a sikert visszajeleztem.

8.5 Motiválás az informatikai gondolkodásra

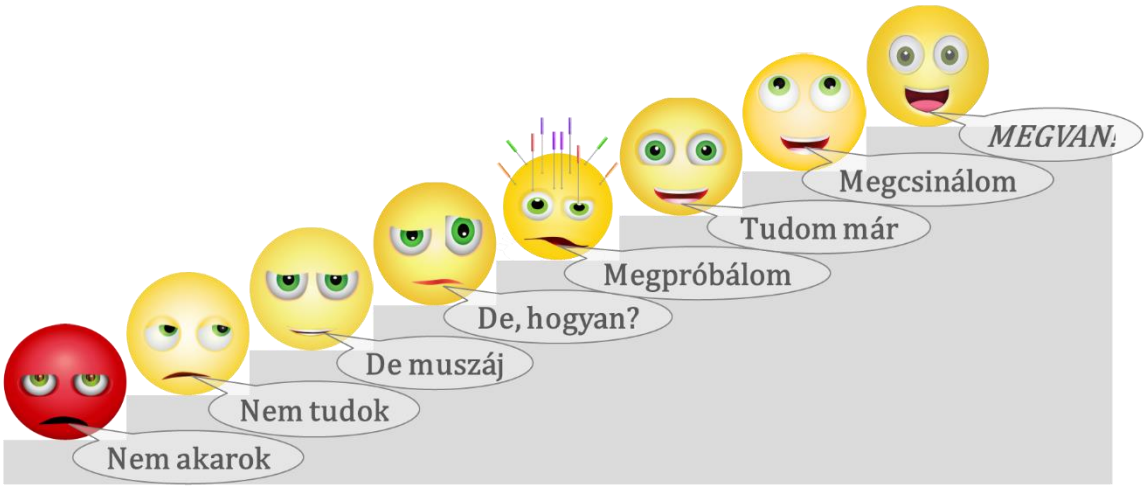
Távoktatásban határozottan kimutatható volt, hogy nem tanítani kell a programozást, hanem a diáknak tanulnia, a tanárnak pedig – Ken Robinsont idézve [142] – **motiválni**, **felszabadítani**, **elvárni** és **önállósítani** kell a diákot. A programozásbéli problémák megoldására Lénárd Ferenc kilenc állapota jellemző [33]: 1. ténymegállapítás; 2. a probléma módosítása; 3. megoldási javaslat; 4. kritika; 5. mellékes mozzanatok említése; 6. csodálkozás, tetszés; 7. bosszankodás; 8. lelkesedés; 9. a munka feladása. A programozás tanulását – ami szintén egy probléma megoldása – egy mémként terjedő motivációs infografika (15. ábra) jellemzi leginkább.



15. ábra Mém, a problémamegoldás motivációs szintjei[32]

A feliratokat a programozásoktatás során jellemző megnyilvánulásokra írtam át (16. ábra), mert ezekre kell egy tanórán azonnal reagálni a tanárnak a megfelelő pedagógiai, oktatási módszerrel.

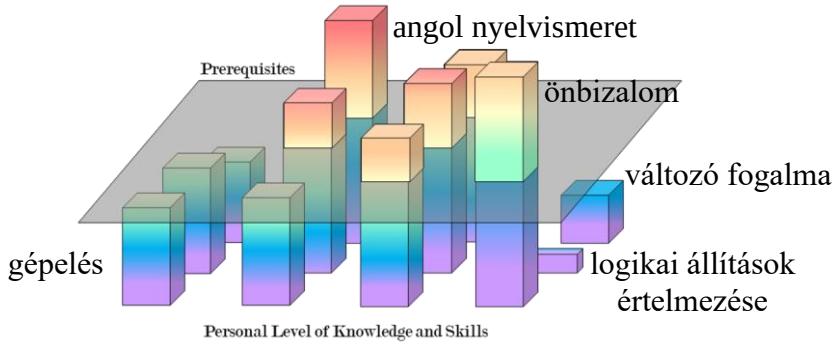
[32] Ismeretlen szerző alkotása: Wich Step Have You Reached Today?. Poster, kép: https://www.scribd.com/document/390586902/Which-Step-Have-You-Reached-Today [2021.10.30]



16. ábra: A programozás tanulásának szintjei[33]

- Pedagógiai szempontból a „Nem akarok” a legtöbb energiát felőrlő szint. Vagy meg kell győzni a továbblépésre vagy meg kell buktatni a diákot (ez is a meggyőzés egy módja). Akarata ellenére senkit sem lehet megtanítani programozni.
- A „Nem tudok” szinten sok beszélgetés szükséges, a lelki, motivációs gátakat le kell küzdeni. Itt – és a későbbiekben is visszatérően – kiemelt tanári feladat a diák kitartásának és hibákból való tanulás képességének fejlesztése. Sue Sentance és Andrew Csizmadia kutatásának [143] eredményét idézve:
  „one specific focus should be on supporting students to develop resilience and the ability to learn from mistakes”
- A „De muszáj” szinten érdemes áttekinteni a programozáshoz szükséges témákat (fejlesztő környezet, algoritmus, adattípus, kódolás, futtatás, hétköznapi logika, szövegértés).
- A „De hogyan” megválaszolásához fel kell mérni az egyes témákban a diák tudását, illetve a feladat megoldásához szükséges ismereteket, készségeket és képességeket. Ki kell emelni az erősségeket, fel kell térképezni a hiányokat. Szükség esetén meg kell határozni azokat az egyéni, kis tanulási lépéseket [144] – beágyazott LAU-okat (17. ábra) –, amikkel a feladat megoldható.

[33] Interneten fellelhető változat átalakítása a programozásoktatás során tapasztalt tipikus megjegyzésekkel. Az ikonok Nina Garman szabadon letölthető képei: https://pixabay.com/hu/users/BilliTheCat-7996303 a Pixabayen (https://pixabay.com) [2021.10.30]



17. ábra: Infografika: a programozáshoz szükséges részképességek (megelőző LAU-ok) térképe[34]

- A „Megpróbálom” szinten a tanárnak „biztosító kötél” szerepe van. A diáknak egyedül kell próbálnia, a tanár – a diák problémája, kérdése esetén – segít a válasz megkeresésében, esetleg válaszol. Fontos megkülönböztetni a „biztosító kötelet” a „mankótól”, mert a mankó csak a feladat megoldásához elegendő, nem teszi lehetővé a megszerzett tudás átélését, a következő szintre lépést. A programozás dolgozatban a valódi „Megpróbálom” jellemzően az elégséges (2) szint.
- A „Tudom már” szinten a diáknak van elképzelése a megoldásról, de még sokszor belezavarodik vagy belefárad a részletes kivitelezésbe. Ezen a szinten kevésbé releváns Pólya problémamegoldási modellje [32]. Segítség kell a helyzet tisztázásához. A pedagógus szerepe a problémák meghallgatása, a megoldás „együtt” végig gondolása. Dolgozatban ezt értékeltem közepes (3) jeggyel.
- A „Megcsinálom” a megoldás tudását jelenti, de előfordulhatnak félreértések, elírások… olyan bosszantó hibák, amit az, aki csinálta nem vesz észre. Idő, gyakorlat kérdése, hogy a diák megtanulja, mit, hogyan szokott elrontani (reláció jelet, pontosvesszőket, 0-tól indexelést, betűcserét, változóneveket). A tanárnak ezen a szinten néha nagyon nehéz feladata van, mint tűt a szénakazalban keresi a hiba okát a diákjának a kódjában. De azért ez a megoldás egy dolgozatban már jó (4).
- A „Megvan” elérése, ha az tényleg önállóan történt, akkor a győztes sikerélményét adja. Valóban sikerült legyőzni, megoldani a problémát, „megszülni” a megoldást, létrehozni valamit. A tanár feladata: örülni és jelest (5) adni, továbbá – kis idő elteltével – újabb kihívásokat tűzni a diák elé, hogy azt is teljesíthesse.

[34] Saját készítésű illusztráció

9 Összegzés

Kutatásom az informatika (programozás) oktatásának módszertani kérdéseiről szólt. A célom az volt, hogy a tudományos elméleti ismeretek és a gyakorlatban alkalmazott szabályzók elemzése és rendszerezése eredményeként ezen ismeretekre építkezve, rendszerszintű válaszokat adjak az informatika oktatásával kapcsolatos kérdésekre.

9.1 Tézisek igazolásának összefoglalása

A megfogalmazott tézisek néhány, a rendszerben kulcsszerepet játszó kapcsolatot írnak le, egymással is összefüggenek.

Tézis: Az informatikatudomány gondolkodási módszereiben és eszközeiben egységes rendszert alkot, amelynek sikeres oktatásához tudományspecifikus oktatásmódszertan szükséges. T1

A 4.7 fejezetben megadott értelmezés mellett az informatika gondolkodási módszereiben kulcsszerepet játszik az informatikai gondolkodás, az adat értelmezése, a modellezés, az absztrakció, a problémamegoldás. Ezekhez kreatív, alkotó gondolkodási tevékenység szükséges, amelyek a tanuló egyéni aktivitásában, „gyakorlati" gondolkodási tevékenységében valósítható meg. A 7. fejezetben elemeztem az alkalmazott módszereket, a 8. fejezetben bemutattam a megfelelő oktatási módszerek alkalmazását. Tanári oldalról a preferált módszer a mentorálás, tanulói oldalról az önálló alkotás. Az előadásnak – mivel a gondolatot nem befogadni, hanem feldolgozni és reprodukálni is kell – csak a rövid változata alkalmas az informatika oktatására. Az oktatás eszközei tekintetében az „IKT eszközök" használata alapvető kell(ene) legyen. Kódolást tanítani és számonkérni csak az informatikai gondolkodás struktúrájának modellezését lehetővé tevő eszközön szabad(na). A papírra írt kód és terv vagy túl vázlatos, ezért más számára értelmezhetetlen, vagy nem gondolkodást tükröz, hanem betanult rutint.

Tézis: Minden, informatikatantervben előírt témát informatikatudományi megközelítéssel oktatva együtt fejleszthető az informatikai gondolkodás, az alkalmazói készségek és a programozási készségek. T2

A 8.1 fejezetben az informatikatudományi megközelítést beépítettem az informatikát oktató tantárgy minden témakörébe. Az informatikai gondolkodás előtérbehelyezésével „minden alkalmazói órán tanítottam programozást is". Ez tanítási szemléletben és a gyakorlatban is jelentős hatással van a digitális írástudás oktatására. Az oktatás során a 7. fejezetben tett megfontolások alapján a 8.2 fejezetben leírt módszereket alkalmaztam, a diákok önálló gondolkodását,

kreativitását támogató mentorálással. Ennek tanítási gyakorlatban igazolt eredményei többek között a 8.3 fejezetben leírt egyedi esetek és a programozási készséget (hajlandóságot) felmérő dolgozat (8.4) szignifikánsan jobb teljesítése.

Tézis: A LAU alapú leírás, valamint ezek kombinációjaként a LAU-modell egy eszköz a tudás-, a készség- és a képességelemek, illetve a tanulási és tanítási folyamat leírására, jellemzésére. T3

A 3. fejezetben definiáltam a Learning Activity Unit-ot, azaz a LAU-t, a tanulás folyamatának, illetve az egyes ismeretminőségek jellemzésére. Igazoltam, hogy ez a modell több neveléstudományi ismeret és tanulási modellel összhangban van, ugyanakkor a tanítás megszervezésekor kihagyhatatlan időbeliséget, folyamatot is értelmezi. A LAU-modell az egyes LAU-ok egymáshoz való viszonyát, kapcsolódását írja le, modellezve az ismeretek és készségek közötti kapcsolatokat is. A 5–8 fejezetekben rendszeresen használtam a LAU-t a tudás és készség tanulási állapotának vagy változásának jellemzésére. Ezzel elkerülöm, hogy a „tudja”, „ismeri”, „érti” fogalmakat szinonimaként használva képlékeny legyen az elvárt, tapasztalt vagy célként megjelölt ismeret, illetve készség. A LAU használatával leírt megállapításokat felhasználtam a tananyag felépítése, a tanítás tervezése, a mérés-értékelés követelményeinek meghatározása és diákjaim tudásának jellemzése során is.

Az 6. fejezetben a tanterv vonatkozásában, a 7. fejezetben a tananyag tartalmában, a 8. fejezetben az oktatási módszerek tekintetében kimutatható, hogy az informatikai gondolkodás fejlesztése tipikusan kapcsolódik egy-egy adott ismeret ᴸ5c, Create jellegű felhasználásához.

Tézis: Az informatikai gondolkodás – ezzel együtt a programozás – képességének a fejlesztését és gyakorlását a motiváció, az érzelmek, a mentális állapot katalizátorként segíti vagy blokkolja. T4

Az informatikai gondolkodás és programozással kapcsolatos ismereteket az informatika oktatás témaköreibe beépítve részleteiben is vizsgáltam, hogy mi okozza a kreatív, problémamegoldó és informatikai gondolkodással, illetve a programozással kapcsolatos problémákat. A 8.1 fejezetben bemutatott példákon kívül is tapasztaltam, hogy az ismeret vagy készség meglétének ellenőrzéséhez a problémás esetekben először a diák hozzáállását, az ismerethez való viszonyát kell megváltoztatnom, azaz a motiváció elsődleges, a különböző (külső és belső forrásokból táplálkozó) félelmek erősen gátolják a feladat megoldását [33]. A programozás tanítását a megfelelő hozzáállás elérésével kell kezdeni, azzal, hogy a diák meg akarja oldani a feladatot, akarjon kitalálni, rájönni, alkotni egy megoldást. Ezt szemlélteti a 16. ábra: A programozás tanulásának szintjei. A pozitív hozzáállás mellett nem találtam akadályt az informatikai gondolkodás

és problémamegoldás területén, legfeljebb a gyakorlatlanság miatti bizonytalanság okoz lassúságot.

9.2 Tapasztalatok

Az informatikai gondolkodás, az adat értelmezése, a modellezés, az absztrakció és a problémamegoldás kreatív, alkotó gondolkodási tevékenységként valósul meg, továbbá e tevékenység végzése a korábbi ismeretektől, motiváló és gátló érzelmi tényezőktől is függ, ezért minden megoldás másképp valósul meg. Ebből következően a tanulási utak is eltérőek lesznek, amelyek sikeres mentorálásához alkalmazkodni kell az oktatási módszerekben. A programozástanítás módszerei [150] közül – amikor mindenkinek tanítjuk a programozást – tanári oldalról a probléma alapú módszer a legalkalmasabb, mert ez az összes többi módszer felé is nyitott. Az a sikeres módszer, ami a tanulóhoz alkalmazkodik (personalised education, személyre szabott oktatás, tömegesen egyéni oktatás). Az egyetemi szakokon a programozásoktatás módszertana szakmaspecifikus, a továbbtanulás irányát ennek tudatában célszerű kiválasztani.

Munkám és kutatásom során az informatikaoktatás személyes megéléséből hatványozottan részesültem. A megélt vagy látott helyzetek az oktatásra, az informatikaoktatásra, a személyiségemre vonatkozó tapasztalatokat eredményeztek. Átéltem, átéreztem, levontam a magam következtetését. Disszertációm fókuszában oktatók és tanulók, azaz mentálisan érzékeny emberek állnak. Az eredményekhez az alábbi megfigyelések is hozzátartoznak.

- A programtervező informatikus és mérnökinformatikus gondolkodásmódja közötti különbség a mindennapok szintjén sok vita forrása. A programtervező rendszerben képzett tanárt lelkileg, szakmai identitásában sérti az a diák, aki mérnökinformatikustól tanult informatikát. A tanárképzésben mindkét gondolkodásmódot, a gondolkodásmódok közötti rugalmas váltást tanítani kellene, sőt, ezt a közoktatásban is érvényesíteni kellene.
- A mérnökinformatikus és programtervező gondolkodásmódbeli és problémamegoldási stratégiabéli különbség sokszor rivalizálást is eredményez. Ez mind tudományos, mind az emberi (munka) kapcsolatok szempontjából káros. Az ellentétes nézőpontok szükségesek a teljes kép kialakításához, amelyben a két nézőpontnak egyenrangú szerepe kell legyen.
- Lassan közhely, hogy az IQ mellett egyre nagyobb szerepe van az EQ-nak. EQ deficites környezetben mediációval hatékonyan tud kompenzálni egy magas EQ-val rendelkező egyén, de eközben rendkívül elfárad, kiég.
- A tanár, az oktató személyiségének egyik legfontosabb alkotóeleme, munkája értékének alapja a szakmai hitelesség. Hosszútávú, akár a magánéletre (sőt az életre) is kiható káros

hatása lehet, ha ezt akár egy rákényszerített tananyag, akár a „jó szívére” apelláló minősítés elvárása rombolja.

- Az informatikatanárt szakmai ismeretei képessé teszik arra, hogy az ebben partner tanulóknak informatikai gondolkodást, a programozást megtanítsa. Az a tanuló, amelyik nem partner, az jelen van, de nem vesz részt a tanulásban; ennek megfelelően kell eljárni.
- A teljesítmény értékelése, minősítése objektív, nem üzleti alku tárgya és nem vegyíthető az értékelt emberi minősítésével, ami szubjektív. Az ilyen irányú törekvéseket csírájában kell elfojtani, ha lehet, rendszer szinten kell megakadályozni a szakmai hitelesség ilyen formában történő támadását.

– Az információs társadalomban alapvető kérdés, hogy ki vagy mi, kit vagy mit vezérel. Az informatikaoktatás akkor sikeres, ha a diákok megtanulják vezérelni a gépeket, eszközöket és szoftvereket [37], ugyanakkor megtanulnak szabadok maradni, nem engedik, hogy a gépek, eszközök vezéreljék őket. Értendő ez a számítógépes játékok életvitelre gyakorolt hatására épp úgy, mint a véleménybuborékok kezelésére, a gondolatainkat vagy állapotunkat leső intelligens szolgáltatásokra, MI asszisztensekre.
– Az informatikai gondolkodás kreatív tevékenység. A kreativitásnak, a jó ötlet kigondolásának az időbelisége bizonytalan. A problémamegoldás készségének fejlesztése egyben a személyiség megismerésével és fejlesztésével is jár. A sikertelenségtől való félelem legyőzése, a tévúton járás bizonyossága majd csalódás kezelése, a tehetetlenség érzés dühe, tudnom kellene görcse… a problémamegoldás velejárói [33], melyeknek kezelése és legyőzése egyéni technikájának kialakítása fejlesztési cél.
  - A jó tanárnak tudnia kell azt, amit a tanuló tanul, a tudásért ő is megküzd.
  - A jó tanár nem a megszerzett tudást adja át, hanem mentori szerepében támogatja, hogy a diákja is megszerezze a tudást. „Motivál, felszabadít, elvár, önállósít” [142].

Mellékletek

SZALAYNÉ TAHY ZSUZSANNA

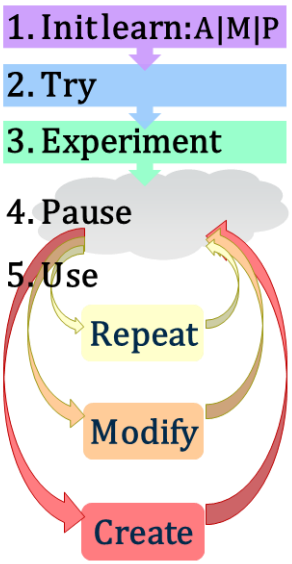
AZ INFORMATIKA (PROGRAMOZÁS) OKTATÁSÁNAK
MÓDSZERTANI KÉRDÉSEI
DOLGOZATÁHOZ KAPCSOLÓDÓ
FORRÁSELEMZÉSEK, ESETTANULMÁNYOK, HÁTTÉRTANULMÁNYOK,
IDÉZETEK, PÉLDÁK, MINTÁK

I. Learning Activity Unit fázisai

A LAU leírása és ábrázolása – igazodva az informatikában használatos kifejezési módhoz – az algoritmizálás, illetve az állapotgép leírásának elemeit tartalmazza, amely az átmeneteket és időbeliséget hangsúlyozza (18. ábra).

1. **Initial learn**: a tananyag megismerése. A befogadás minősége:
   A) **Active** megismerés: az új ismeret azonnal használható.
   M) **Moderated** megismerés: az új ismeret kapcsolódások mentén visszakereshető.
   P) **Passive** megismerés: az új ismeret nincs kontextusba más ismeretekkel.
2. **Try**: kipróbálás, a megértés ellenőrzése.
3. **Experiment**: az ismeret részleteinek kifejtése.
4. **Pause**: szünet (szűrés, felejtés).
5. **Use**:
   a. **Repeat** az ismeret **ismétlése**.
   b. **Modify**: **módosít**ott, tipikus helyzetekben felhasználás.
   c. **Create**: **alkotó**, kreatív felhasználás.

← Újra a 4. pont, vagy újra az 1. ponttól.



18. ábra: A LAU fázisai

Egy LAU időben több éven át, akár élethosszig tarthat. Például, ha egy programozási nyelvet valaki megtanul és használja nap mint nap, akkor egy 5c fázisú tudásról beszélhetünk. Azonban, ha ezután évekig nem használja az adott nyelvet, akkor ez a tudás erősen elkopik, idővel 5a szintre csökken. Sőt, az is előfordulhat, hogy célszerűbb újratanulnia, ami valószínűleg egy 1A jellegű fázis lesz. Egy LAU 5c fázisa lehet például extrapoláció, asszociáció.

A LAU a tanulási folyamat elnagyolt, egyszerűsített, ezzel együtt pontatlan, vázlatos leírása. A témával kapcsolatos tudományos kutatások eredményeit intuitíven kombinálja. Célja az oktatásmódszertani és tanulásmódszertani elméletek szintézise a gyakorlat szintjén, a tanulásról szóló megállapítások közös „nyelvének” megteremtése.

I/1. Initial learn

A fázis neve Ebbinghaus felejtési görbénél [24] jelenik meg. A tudásról szóló modellek az ismeret mélységét vizsgálják, azaz egy állapotot és nem azt, hogy hogyan jut a tanuló abba az állapotba. A felejtési görbe igazolásánál komoly nehézség volt, hogy mit, azaz a tudás melyik állapotát méri, mennyire igazak az állítások más állapotok esetén, mennyire igaz két állapot

közötti átmenet esetén. Ezen kérdések vizsgálata még nagyon sok doktori disszertációt eredményezhetnek, azonban tanítani addig is kell. Ezért azt feltételezzük, hogy egy ismeret valamilyen formában a tanuló birtokába jut. Ezt a folyamatot minősítjük aszerint, hogy az ismeret a birtokló számára mennyire releváns, mennyire tudja korábbi ismereteivel kontextusba helyezni. A kezdeti tanulás:

- aktív, ha szervesen kapcsolódik a korábbi ismeretekhez, hiánypótló.
- passzív, ha nem kapcsolódik semmihez, korábbi tapasztalattól függetlenül meg kell jegyezni vagy – például sokszori ismétlődés miatt – tudatos tanulás nélkül rögzül.
- moderált, ha léteznek korábbi ismeretek, de a kapcsolat ezekkel nem nyilvánvaló.

Példa: „T45 jelű tanulócsoport névsorának megtanulása”

Ha a csoport tagjai gólyák, akkor a tanulás valószínűleg passzív módon lehetséges. (Nevek tanulása a csoporttagok ismerete nélkül.) Ha egy régóta ismert közösségből származnak a csoporttagok, akkor moderált a névsor tanulása. (A csoporthoz tartozás tulajdonságot kell ismert emberekhez hozzárendelni.) Ha pedig kiderül, hogy egy korábban ismert csoportról van szó, csak a „T45” csoportnév új, akkor aktív az ismeretszerzés.

A minősítés nem mond semmit a tanuló magatartásáról, szorgalmáról és a tanulás módszeréről sem, de ezek – az előismereteken túl – befolyásolhatják az eredményt. Másik oldalról nézve, a tanítás során a tananyag megfelelő felépítésével – LAU-ok időbeli elrendezésével – törekedhetünk arra, hogy egy ismeret megtanulásának első lépése aktív legyen, de nem hiba, ha passzív ismeretszerzést várunk el a tanulótól. Sőt, sok esetben a reklámok, tájékoztatók, információs táblák a passzív ismeretszerzésre építenek és a tanítási gyakorlatban is jellemző, hogy egyes fogalmak bevezetését jóval a tényleges tananyagbéli ideje előtt kezdjük, mintegy megemlítve azt, hogy ezzel motiváljuk a megismerés további lépéseit, kíváncsiságot ébresztve a tanulóban.

I/2. Try

A kipróbálás fázisa általában rövid, akár egybe is lehetne venni az 1. fázissal, sőt, néha meg is előzheti azt. Külön említése azért indokolt, mert problémát okoz, ha kimarad. Aktív tanulás után a kipróbálás lényegében egy gyors ellenőrzést jelent. Passzív ismeretszerzés esetén fontos információ, hogy a begyűjtött új ismeretnek van-e valami értelme. Moderált ismeretszerzés során nem kérdés, hogy van-e értelme, de kérdés, hogy jól gondoljuk-e, jól csináltuk-e, használható-e az új ismeret.

Az ismeretszerzés és kipróbálás között nem telhet el sok idő. Lényegében azonnal szükséges ez a fajta megerősítés, ami hosszabb előadások során előadói demonstrációként is megvalósulhat, írott szövegben pedig példaként jelenhet meg. A tanulás megszervezésekor azonban figyelni kell arra, hogy ezek csak motiválnak a kipróbálásra, a valódi Try során a tanulónak kell kipróbálnia, használatba vennie az új ismeretet.

Az az ismeret, amit nem próbálunk ki bizonytalanság érzést, félelmet kelthet, vagy rövid időn belül elfelejtődik. Például, ha az új ismeret „bemutatkozáskor a másik fél neve", akkor azt rögtön használva, visszamondva „kipróbáljuk". Ha ez elmarad, nehéz lesz visszaemlékezni, mit is mondott partnerünk; a visszaidézés során elbizonytalanodást élünk meg; a tévedéstől és ennek következményeitől való félelem érzése hatására inkább kerüljük a néven történő megszólítást.

I/3. Experiment

Ebben a fázisban történik az új ismeret részleteinek elemzése, jellemzése, megismerése; az ismeret kibővítése, kapcsolása a korábbi ismeretekhez. Passzív ismeretszerzés esetén nagyon fontos fázisa a tanulásnak, de aktív ismeretszerzés után is hasznos. Fontos, hogy a kipróbálást követően kezdjünk el belemenni a részletekbe, mert a vizsgálódáshoz valós tapasztalat szükséges. Egy elméletileg létező, használható dolgon elméleti módosítást végrehajtva elméletileg juthatunk valamilyen eredményre, de ez nem lesz valós dolgokhoz, problémákhoz kapcsolva. Például: meg lehet tanulni a szorzótáblát verselve, a szorzás műveletének (többszörös összeadás) kipróbálása nélkül, de az eredmény az csak egy visszamondható szöveg lesz, nem számolási tevékenység.

Az Experiment fázis az ismeret rögzítését – kimondása, leírása, strukturálása, használata – jelenti, de egyben kibővítésre, azaz új ismeret szerzésére is van mód. Emiatt ezen fázison belül megjelenhet beágyazva más tanulási egység (LAU) is. Ha az ismeret bővül, akkor arra is szükséges kipróbálás, a diszkusszió minden ágára kell példa.

A tudás modellekben, ez a fázis a befogadás, a sajáttá tétel utolsó fázisa. Sok esetben tapasztalhatjuk, hogy a tanulási folyamatnak itt vége. Ha tényleg ezzel fejeződne be a tanulási folyamat, akkor egy feleslegesen, önmagáért megtanult dologról lenne szó, ami sohasem került gyakorlati felhasználásra és ezért idővel el is felejtődik.

I/4. Pause

Egy tanulási egység tanulása nem folytonos. A tanórák (és élethelyzetek) váltják egymást, a témakörök mindig újak. A tanult ismeret akkor hasznos, ha idővel fel tudjuk használni. Idővel,

azaz valamennyi szünet után. A tanulás során fellépő szünetek szerepe a tanulásmódszertan egyik örök témája. A felejtés jelensége; a felidézés nehézsége, illetve segítése; a rövidtávú és hosszútávú memória működése – mind kutatási területek, bőséges szakirodalommal és többé kevésbé ezek alapján hirdetett módszertani tanácsokkal. A megfelelő tanulási módszer kiválasztásakor az ismeret jellege mellett nagyon fontos szempont az is, hogy milyen hosszú a szünet az ismétlések között. Fontos szempont, hogy nem létezik felelevenítés szünet nélkül, ahogy az is fontos, hogy a túl hosszú szünet után nincs mit feleleveníteni.

I/5. Use

Az ismeret használata alapvető feltétele annak, hogy tartóssá váljon. A magyar közoktatás régi problémája, hogy a poroszos rendszert alkalmazva, a megtanításra, az ismeretek megjegyzésére helyezi a hangsúlyt, míg az információs robbanás hatására a nemzetközi trendek és a magyar szakmai és piaci igények is a kompetenciákat, azaz az ismeretek felhasználási képességét helyezik a középpontba. Ezért a tanulás folyamatát és eredményességét nem elég az ismeret átadásáig nyomon követni.

a) Repeat

Az ismeret felhasználásának legegyszerűbb módja, ha a tanult formában kell használni. A tanuló felmondja a leckét, visszaidézi a tanultakat. Ezt a fajta használatot várja el általában egy kisdolgozat, egy beugró, egy órai felelet. A feladat jellemzően konkrétan rákérdez az adott ismeretre.

b) Modify

Nagyobb kreativitást igényel a felhasználás, ha a tanuló tudja, hogy az ismeretet fel kell használnia, de a felhasználás módját neki kell kitalálnia. Jellemzően nagydolgozatok, témazárók feladatainál találkozunk ilyen elvárásokkal. A feladat specifikálása adja meg, hogy melyik ismeret felhasználását kell bemutatni. A használandó ismeretet egy, a témára jellemző néhány elemű listából kell kiválasztani, kreativitást csak a felhasználás módja – például az elemek kiválasztásának helyes sorrendje – igényel.

c) Create

Az ismeret készségszintű felhasználása azt jelenti, hogy külön instrukció nélkül kerül felhasználásra, amivel a megoldáshoz közelebb lehet jutni. Tanulási helyzetben, projekt során felmerülő feladatokban, illetve egyes „mindent lehet használni” dolgozatokban lehet megfigyelni az ismeret ilyen szintű felhasználását. Ebben az esetben az eszköz kiválasztása és az eszköz felhasználásának a módja is szabadon választott, a kreativitás mértékét különböző hatékonysági,

célnak megfelelési szempontokkal lehet jellemezni. Nem jelent valódi kreativitást, ha a tanuló minden helyzetben ugyanazt az ismeretet választja, mert ez valójában egy egyéni specifikáció, annak az egy ismeretnek a módosított felhasználása. Sokszor a tanuló maga számol be arról, hogy egy ismeretet valós helyzetben, alkotó módon használt, mert ez sikerélményt nyújt. „Elmagyaráztam a testvéremnek", „…és akkor eszembe jutott, hogy ez talán jó lesz", „és akkor rájöttem, hogy amit régebben csak úgy beírtam, az ott nekem miért jó". Ehhez hasonló „megvilágosodások" bizonyítják, hogy az ismeret valóban, kreatívan használható a tanuló számára.

A felhasználás aleseteinél meg kell különböztetni a használathoz szükséges felhasználási módot és a tanuló képességét. Ha az Initial learn minősége aktív volt, akkor lehet, hogy azonnal kreatívan tudja használni a tanuló az ismeretet, de nem lesz gondja az alacsonyabb szintű felhasználási móddal sem. Ez azonban nem jelenti azt, hogy a tanuló használat nélkül, néhány héttel később fel tudja idézni akár ismétlésre, akár kreatív módon az ismeretet. Megfelelő feladatokkal idővel elérhető a magasabb szintű használat, ugyanakkor használat nélkül az idő múlásával kevésbé valószínű, hogy egy adott helyzetben „beugorjon", hogy az ismeret használható lehet. A használat részletei is elveszhetnek a hosszútávú memória süllyesztőjében. Ha a használathoz a felidézés nem sikerül, akkor célszerű újratanulni, azaz egy új LAU-ként beiktatni az adott ismeretet.

3.1.1

II. LAU-modell

Kutatásom során, ahhoz, hogy érdemben beszélni tudjak a tanulás folyamatáról, a folyamatban fellépő akadályozó tényezőkről és problémákról, szükségem volt egy, a folyamatot leíró modellre. Az angol nyelvű megnevezést a szakirodalomra utalások miatt választottuk a tanulás absztrakt leírására, amely egy tananyagelem, fogalom vagy témakör (learning unit) tanulásának folyamatát (learning activity process) írja le. A Learning Activity Unit egy tanulási sablon tantervek, tanmenetek minősítésére, tanítási és tanulási célok megfogalmazására, a tanulási folyamat során felmerülő problémák detektálására, illetve a kimenet (az elvárások) pontos meghatározására. A LAU-modell (angolul LAU-Model) a LAU-ok (egységek) egymáshoz kapcsolódását írja le.

Példa unitok kapcsolatára:

Ha megtanulandó a számlálós ciklus készségszintű kódolása egy adott nyelven, ennek praktikusan szükséges előismerete az értékadás alkalmazásának ismerete.

Példa probléma detektálásra és megoldási tervre:

LAU = for-ciklus kódolása. A ciklus készségszintű kódolása nem alakul ki, ha kódkiegészítést használ a tanuló. Amennyiben a számonkérés papíron lesz, a kódolás nemcsak gépelve, hanem kézírással is készségszintű kell legyen. Probléma kezelése: előismeret a gépelés és a kézírás ismerete, de emellett figyelni kell a gyakorlófeladatok specifikálására és a gyakorláshoz szükséges időre is.

A LAU és LAU-modell használata lehetővé teszi a példában szereplő állítások gyors és pontos megfogalmazását, amellyel a tanítási helyzetekre jobban fel lehet készülni; „futási időben”, azaz a tanórán több, a helyzethez jobban alkalmazkodó döntést lehet hozni. Magasabb szinten a LAU és LAU-modell alkalmas eszköz tantárgyfejlesztéshez, tematikák elemzéséhez, mérés-értékelési módszerek kidolgozásához, minőségbiztosítási elemzésekhez.

II/1. LAU-modell formális leírása

A LAU-modell az egyes ismeretek tanulásának kapcsolódását írja le. Bár a felhasználás és a gyakorlat szempontjából nem fontos, a LAU és LAU-modell formális leírása is lehetséges, amivel formalizálhatók tipikus tanmenettel kapcsolatos megfontolások. Jelen esetben a formalizálásnak csak a tartalom struktúrájának a bemutatása a célja, de további kutatásokkal akár egy tanmenetgeneráló eszköznek is lehetne az alapja. A LAU formalizálásában az egyes fázisokat a ^{L} jellel különböztetem meg egyéb számozásoktól és jelölésektől.

Példa:

A for ciklus – for (int i = 1; i <= 10; ++i) {…} – tanításához szükséges előismeretek: integer adattípus, értékadás, matematikai-logikai kifejezések kiértékelése, érték növelése, szintaxis elemek (paraméterezés, blokk-képzés, utasítás lezárás). A tanítás célja, hogy a tanuló a számlálós ciklust kreatívan tudja használni programozási feladatok megoldásához. Függvényként megfogalmazva: kell egy T tanulási eljárás:

forLoop^{L}5c T(datatype^{L}3, assignment^{L}5c, expression^{L}3, incrementation^{L}3, syntax^{L}5a)

Példánkban a forLoop, datatype, assignment… egy-egy LAU, amelyek ismeretének szintjét is megadjuk a LAU-fázis jelölésével.

A LAU fázisokat érdemes a 0. fázissal – nem hallott még az adott ismeretről a tanuló – kiegészítve egyértelműsíthető, ha a LAU-nak csak egy részletéről szeretnénk beszélni. Ezzel a kivitelezéskor meg tudjuk különböztetni, hogy – a példánál maradva – a forLoop tanulási egységnek a kezdetétől (0-ról) vagy valamilyen továbbfejlesztés formájában gondolkodunk. A folyamat egy részét zárt intervallumaként adhatjuk meg. Például forLoop^{L}0-5c a teljes folyamat, forLoop^{L}1-3 vagy forLoop^{L}5a-5b lehet egy-egy tanórára tervezett rész, illetve jellemezhetjük a tanuló fejlődését, forLoop^{L}1A-5c jelölésekkel, esetleges visszafejlődését a forLoop^{L}5c-5a jelöléssel.

A függvény törzsében tanári és tanulói tevékenységek lehetnének, ami kellő tanári tapasztalattal tervezhető, majd a tanítás-tanulási folyamatban megvalósul. Azonban a tanítási gyakorlat nem ennyire egyszerű. A legtipikusabb problémák az előismeretek változatos minőségéből erednek. Azaz a kivitelezésnél nagyjából arra számíthatunk, hogy az előbbi példában inkább datatype^{L}?, assignment^{L}?, expression^{L}?, incrementation^{L}?, syntax^{L}? írandó, ahol a kérdőjel helyére tanulónként(!) az adott pillanatban jellemző érték helyettesítendő. A tananyag készítésekor figyelembe kell venni a várható értékeket, a tanítás-tanulás során fel kell ismerni, hogy milyen szintű valójában a tanulók tudása és kezelni kell a tanulócsoporton belül az értékek szórását (differenciált oktatás).

A LAU-modellben az egyes LAU-ok lehetnek egymás után szervezve vagy egymásba ágyazva, illetve lehet párhuzamosan szervezni az egységeket. A tananyagtervezésnél ezek az eljárások teljesen megszokottak, azonban a LAU sajátossága, hogy az időt is figyelembe veszi, azaz az előzetes ismereteknél nem azt kell figyelni, hogy előzetesen tanulta-e a tanuló, hanem azt, hogy az előzetes tanulmányaiból mennyi maradt meg a felhasználás idejére. A LAU-modell már a tervezés során támogatja a differenciált, egyéni igények figyelembevételére alkalmas

megoldásokat is, segít az egyéni tanulási utak tervezésében, megvalósításában. Emellett tanórai akut helyzetekben támogatja a tanítási-tanulási problémák kezelését.

II/2. A LAU-modellel leírható tananyagfelépítés általános törvényszerűségei

1. Egymás után szervezett LAU-okra tipikus példa a tanév során egymás után tanult, egymáshoz nem kapcsolódó témakörök sorozata. Jellemző probléma, hogy a nem használt tudás idővel kopik. Tipikus eset, hogy A_LAU^{L}0-5b majd B_LAU^{L}0-5b után A_LAU^{L}3 fázisra lehet számítani. Egymásra épülő témák esetén a későbbi tanulási egységbe célszerű belevonni a korábbi egység fejlesztését is: A_LAU^{L}5a-5c (B_LAU^{L}0-5b, tavalyiA_LAU^{L}0-5b).
2. Egymásba ágyazott LAU-ok esetén figyelni kell arra, hogy a beágyazott LAU jóval kisebb egység legyen (bonyolultságban, időben), mint amibe beágyazzuk. Azonos vagy nagyobb méretű LAU beágyazása – főleg a LAU^{L}0-5a folyamatba – azzal a veszéllyel jár, hogy a tanuló elveszíti a fonalat. A jelenség hasonló egy hosszú körmondathoz, amiben témaváltás történik és a kitérő nagyobb hangsúlyt kap, mint a célirány. Pedagógiai szempontból fontos eset, amikor kiderül, hogy a tanuló azért nem jut előre egy A_LAU folyamatában, mert hiányos a tudása az (esetleg több) előismeret terén. A hiányosságok pótlása az A_LAU felfüggesztésével, az előismeret LAU beágyazásával lehetséges, de sokszor célszerűbb az előismereteket tényleg az A_LAU elé helyezni, a pótlás után elölről kezdeni az A_LAU-t.
3. Ha egy X_LAU megtanulásához sokféle előzetes ismeretre van szükség, akkor hasznos lehet az egyes LAU-okat párhuzamosan venni. (Például egy színdarabhoz a szöveget, színpadi mozgást, betétdalokat lehet párhuzamosan tanulni.) Azonban a párhuzamos tanulás jellemzően a részfolyamatok egymásutániságát jelenti, azaz, például A_LAU^{L}0-1, B_LAU^{L}0-1, C_LAU^{L}0-3, A_LAU^{L}1-2, B_LAU^{L}1-3, A_LAU^{L}2-5b, C_LAU^{L}3-5a, B_LAU^{L}3-5b. A LAU-ok párhuzamos tervezésnél figyelni kell a megfelelő időosztásra, minden LAU-ra úgy kell visszatérni, hogy gyorsan fel lehessen idézni az utolsó állapotát, azaz el kell kerülni, hogy a ≪bármelyik≫^{L}4 károsan hosszú legyen. A párhuzamosságnak még két buktatója van: ha túl sok LAU van párhuzamosan, akkor követhetetlenné válik a tanulás, illetve hasonló LAU-ok könnyen összekeveredhetnek.

Példák párhuzamosan szervezett LAU-ok problémáira:

- Egyszerre tízféle tantárgy tanulása sokkal könnyebb, mint tizennégyféle tantárgyból követni a tananyagot, leckét, határidőket.

- Sokaknak okoz problémát egyszerre két idegennyelv tanulása, például azért, mert ugyanahhoz a magyar szóhoz egyszer az egyik, máskor a másik nyelv szavát kell párosítani.
- A heti egyórás tantárgyaknál egy tanóra elmaradása általában két tanóra veszteség a felejtés miatt. A sok más tantárgy, program miatt a LAU4 túl hosszú, a visszalépés jelentős.

3.1.1
6.3

III. Pedagógia – szakmai alapok analízise

Már készen volt a LAU-Modell, amikor elgondolkodtam azon, hogy honnan származik az, ahogyan én gondolkodom a tudásról, tanulásról, oktatásról. Nincs pontos emlékem arról, hogy mit, hol tanultam. Egyes részeket valószínűleg egyetemista koromban tanultam, másokat tanfolyamokon, és biztosan szó volt róla pedagógusszakvizsga-képzésen is. Ráadásul magyar nyelven tanultam mindent, sőt, sok mindent még a „szocializmus" idején, így csak példaként tudom megadni, hogy nemzetközi szakirodalomban hol lehet megtalálni azokat az ismereteket, amik számomra természetesek. Az utolsó tankönyv, amiből a pedagógus szakvizsgára tanultam, a Nemzeti Tankönyvkiadó által 1998-ban kiadott Didaktika – Elméleti alapok a tanítás tanulásához, Falus Iván szerkesztette tankönyv [11]. Ebben a projektmódszerről Falus Iván ismertetője két oldal (306–308), Kotschy Beáta az óravázlatot a hagyományos formában írja le, majd kiegészítésként megemlíti, hogy a projektoktatás más típusú tervezést igényel (p. 485–486.). Amikor tanultam, ugyanennek a tankönyvnek másik szerzőjétől, Nádasi Máriától 2007-ben, akkor már a projektmódszer volt a fő téma és azóta még tizenpár év eltelt. Már Nemzeti Tankönyvkiadó sincs, az egyetemi tankönyvkiadás átalakult. Forrásként a ma interneten megtalálható tankönyvekre, tananyagokra tudok hivatkozni. A munkám során a gyakorlatban, illetve az akkreditált képzéseken tanult ismeretek a pedagógia, a neveléselmélet, az oktatáspszichológia, a tanulásmódszertan diszciplínák területéhez tartoznak. Az informatika oktatásának módszerei számomra ezen ismereteknek – az informatika (programozás) módszertani ismeretei és az oktatásmódszertani ismeretei – az együttese. Lényegében ezen területeket ötvözi az Informatika szakmódszertan napjainkban érvényes meghatározása [10] is.

III/1. Tudás/ismeret/tanulás modellek

Bloom taxonómiája

A tudás minőségéről nagyon sok tudományos munka született. Talán a legjelentősebb Bloom taxonómiája (19. ábra). Sokan kiegészítették, pontosították, más megvilágításba helyezve vizsgálták, továbbfejlesztették. Bloom taxonómiája a Magyarországon használt pedagógia tankönyvekben és nemzetközi szinten is meghatározó elmélet a tudás, az ismeret mélységének, fajtájának meghatározásához [11].

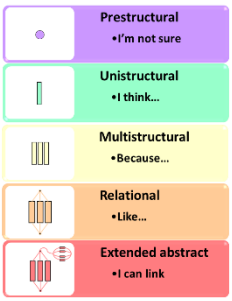
19. ábra: Bloom taxonómiája

Tantervekben, tanmentekben a belépési feltételek és kimeneti követelmények meghatározásakor ezt az osztályozási rendszert használjuk [12]. Valószínűleg ezt az elméletet tanultam vagy

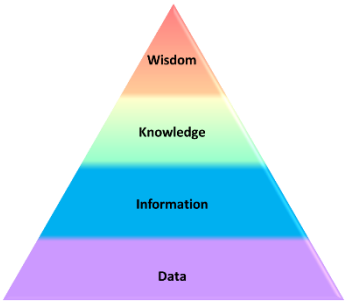
ez hatott leginkább azokra a neveléstudósokra, akiktől tanultam. A Bloom-taxonómia alapján gondolkodom, de a gyakorlatban leegyszerűsítve használom, ami a sokféle megfogalmazás egyvelegéből alakulhatott ki bennem.

SOLO taxonómia

A SOLO-taxonómia [13] eggyel kevesebb szintet határoz meg és gyakorlatiasabb a Bloom-taxonómiánál (20. ábra). A kutatást megelőzően ezt a taxonómiát nem ismertem, de hasonló gondolkodásmóddal jutottam hasonló megoldáshoz. Számomra a jelölésrendszer nem ismerős, de kifejező. Míg a Bloom taxonómiája inkább az ismeret szintjére fókuszál, a SOLO-taxonómia az ismeret felhasználásának a módját jellemzi, ami a tanítási gyakorlatban jobban használható [14].



20. ábra: SOLO taxonómia



21. ábra: DIKW hierarchia

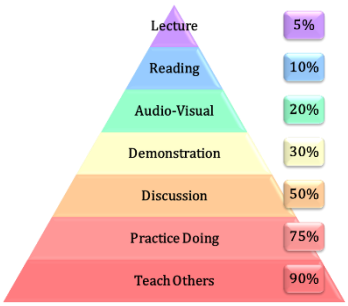
Data–Information–Knowledge–Wisdom hierarchia

Valószínűleg teljesen más irányból közelítve, információelmélettel, rendszerelmélettel kapcsolatosan – így informatikusok körében talán ismertebben – jellemezhető a tudás a DIKW, azaz data–information–knowledge–wisdom hierarchiával [15]. A tudás eme értelmezése is továbbfejlődött: vitatták, kiegészítették [16]. Az eredetileg három szint kiegészült, többféle módon értelmezték az egyes szinteket és a piramissal ábrázolást (21. ábra) [17].

Learning Pyramid

Az eddigi bemutatott modellek a kutatásom és gyakorlati felhasználás szempontjából azért nem megfelelők, mert az ismeretről, a tudásról szólnak és nem a tanulásról. Az egyes szinteket állapotként mutatják be, nem az ismeret megszerzésének folyamataként.

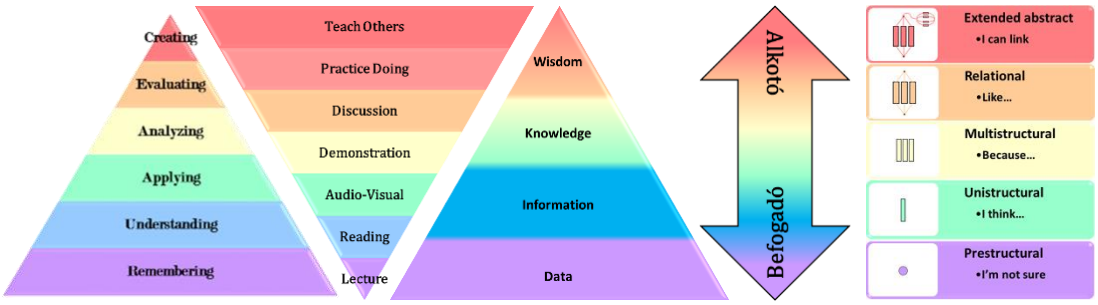
A tanulási módszerek hatékonyságára vonatkozik a Learning Pyramid (Tanulási Piramis) [18], melyet a szakirodalom az alkotója után „Edgar Dales's Cone of Experience”[19] néven is ismer (22. ábra). Sokszor százalékos jelölés is látható mellette, ami elvileg azt jelentené, hogy az adott tanulási módszer mennyire hatékony, a tudás hány %-ára emlékszik a tanuló a tanóra után [20]. Az 22. ábra sokak szerint azt sugallja, hogy mások tanításával lehet legtöbbet megtanulni, és rögtön el is utasítja azzal, hogy egy csecsemő nem tud tanítani. Mások – ahogy a DIKW piramisnál is olvashatjuk – a piramis állását, jelentését, a részek egymásra épülését (egymáshoz képest mutatott viszonyát) vitatják. A Tanulási Piramis talán ezzel a definícióval értelmezhető leginkább: A tanulás hatékonysága (százalékban mért érték) egy adott tevékenység során az egyén által feldolgozott és a kommunikált tananyag hányadosa. Például az egyórás előadás igen kis részét hasznosítjuk, de ha egy órán át beszélgetünk a témáról, akkor az elhangzottak felét fel tudjuk idézni, ha pedig egy órán át tanítjuk, akkor az óra végére tudni is fogjuk, amit tanítottunk. (Másik megfogalmazásban: azt tudod igazán, amit már tanítottál.) Tény, hogy erre nem ismert konkrét kísérlet, mérési adat. Az 22. ábra „jótékonyan” elrejti, hogy mások tanításához először valahogy meg kell szerezni az ismereteket, lehet, hogy a felette lévő módszerek mindegyikét alkalmazni kell, mire felkészülünk mások tanítására. Az sem látszik a modellből, hogy kihagyhatók-e egyes módszerek, azaz, lehetséges-e például, hogy sok olvasással eléggé megtanulva a tananyagot, mások tanítása során mélyülnek és rögzülnek az összefüggések – és a többi módszer tejesen kimarad. Mindezek ellenére, mégis, érezhetően jól rámutat a különböző tanulási és tanítási módszerek alkalmazásának hatékonyságára.



22. ábra: Tanulási Piramis

A Tanulási Piramist csúcsára állítva és Bloom taxonómiájával összehasonlítva azonban egy másik lehetséges értelmezést kaphatunk: a tudás Bloom taxonómiája szerinti minőségét a Tanulási Piramis megfelelő tanulási módszerével el lehet érni [21]. Például, mások tanításakor az ismeret létrehozása, más kontextusban előadása, azaz Creating történik.

A Tanulási Piramis mellett a számokon kívül – egyes ábrázolásokon – még egy skála megtalálható, ami a DIKW hierarchiával való kapcsolatát is mutatja: A tanulási tevékenység a Lecture szinten teljesen befogadó, míg a Teach Others szinten alkotó. A Befogadó-Alkotó felosztás mutatja a korábbiak és a SOLO-taxonómia közötti kapcsolatot is (23. ábra).



23. ábra: Bloom Learning Pyramid DIKW Kiegészítés SOLO

Kitérő – Tudományosság

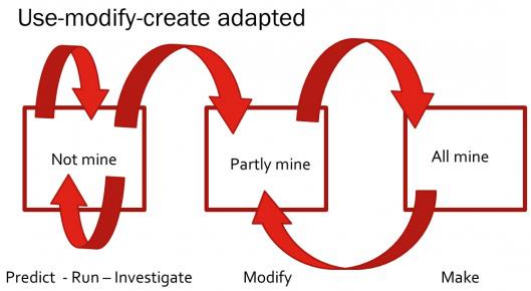
A fentebb olvasható „…százalékos jelölés … elvileg azt jelentené” – megfogalmazás egy matematikus vagy jogász számára elfogadhatatlan. Ki kellene mondani tézisként és be kellene bizonyítani, le kellene vezetni egzakt logikai lépések sorozatával, hogy az eredeti szöveg/ábra ekvivalens a leírt definícióval. A mérnöki tudományok szemszögéből pedig az első kérdés: hol vannak az adatok. Mérésekkel kellene igazolni az állítást.

A Tanulási Piramis sokak szerint tudományosan nem igazolt. A számokra nincs kísérleti adat. A százalékértékek valóban nem mérési eredmények, de „valami van benne”, a mindennapi tapasztalat a relációk irányát igazolja és ezért széles körben elterjedt. A Tanulási Piramis a mindennapi gyakorlathoz, tanulási és tanítási tevékenységekhez sokkal konkrétabb útmutatást ad, mint bármelyik másik modell. Mindegyik modellnél felmerültek kérdések a szemléltetéssel kapcsolatban: a háromszög egyes szeletei azért kisebbek, mert kevesebbek? Vagy a tartalmazást, esetleg a stabil ráépülést szemléltetik? Mindegyik modellnél hosszasan fejtegették nézeteiket a kutatók, elméleti úton tisztázva a modell jelentését. Ezek a piramisok nem mérések alapján absztrahált modellek, hanem infografikák, melyek elvont fogalmak értelmezése, képpé kódolása során keletkeztek.

III/2. Tanulással kapcsolatos egyéb modellek

PRIMM modell

Az angliai informatikaoktatás 2013-ban vett új irányt, minden évfolyamon bevezették a programozás oktatását. A programozás tanulásának jellemzéséhez *Sue Sentance* a programozás tanulásának folyamatát jellemző modellt adott, amelyet a megfelelő kezdőbetűkből PRIMM modellnek [22] nevez. Ebben a modellben találtam egyedül arra mutató jelet (24. ábra), hogy egy tudásszint a következő lépésben nem magasabb szintre változik, hanem marad (not mine → not mine), illetve visszalépés is lehetséges (mine → partly mine) [23]. Erről a publikáció 2017-ben jelent meg, de a kutatásomhoz már 2015-ben szükségem volt egy tanulási folyamatot leíró modellre.

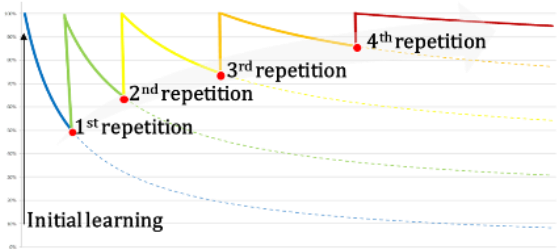


24. ábra: PRIMM modell

Ebbinghaus felejtési görbéje

A tanulás folyamatát jelentősen befolyásoló tényező az idő. Néhány, az idővel kapcsolatos kérdés jól szemlélteti, hogy a tanulás eredményességét a gyakorlatban nem elegendő az ismeret állapotával jellemezni:

- Mennyi idő van a tananyag feldolgozására?
- Mennyi idő alatt lehet elmagyarázni a tananyagot? (Mekkora az adatleadási sebesség?)
- Milyen sebességgel képes tanulni a tanuló? (Mekkora az adatfeldolgozási sebesség?)
- Mennyi idő telik el a tananyag egyes ismétlései, ismételt felhasználásai között? A felejtés vagy a felidézés erősebb-e?
- Mennyi ismétlésre van szükség?
- Mikor lesz újra szükség az ismeretre?



25. ábra: Felejtési görbe

Az egyes tudáselemeknek egy adott pillanatban mutatott minősége semmit sem ér később, ha elfelejtjük. Az adatokból természetesen szelektálunk már az első pillanatban. Vannak adatok, amelyek egy adott helyzetben fontosak, később azonban elfelejthetők. Rövidtávú emlékezet, hosszútávú emlékezet, munkamemória, felidézés, kontextus – ezek a fogalmak megkerülhetetlenek, ha a tanulást, a tudás megszerzését és megtartását folyamatában akarjuk jellemezni. Az ezzel foglalkozó elméletek közös eredete az Ebbinghaus által leírt felejtési görbe (25. ábra).

Ebbinghaus kísérletét reprodukálták és a görbét új mérésekkel igazolták [24], de ezen túlmenően, ezekre az eredményekre is alapozva, az emberi memória „működését” sokrétűen kutatták, elemezték. Számos tézis, antitézis és szintézis övezi, de korántsem mondhatjuk, hogy mindent tudunk az emlékezet működéséről. Azonban az biztos, hogy a felejtés, az adatok szelekciója és az ismeretek ismétléssel történő megerősítése létező jelenségek, ezért a tanulási folyamat modelljéből nem hagyható ki.

3.1.2

IV. A tanulási folyamatot befolyásoló tényezők

A LAU a tervezett tanulás leírásának eszköze, a tanulást cselekvésként értelmezi. De nem minden tanulás tervezett. Lehet, hogy valami olyan dolog történik, amit egy életre megjegyzünk, másrészt van, amikor valaminek meg kellene történnie, a tanulónak észre kellene venni, rá kellene éreznie.

IV/1. Élmény

Nem várt, nem tervezett tanulási helyzet, például, ha valami megcsíp. Utána kiderül, hogy darázs volt-e, allergiás reakciót kivált-e, kell-e keresni a gyógykezelést… Azaz a tanulás első fázisa a Try, miközben elmarad vagy egybeolvad az Experiment fázissal az Initial learn fázis. A tanulásnak ez a formája nagyon inspiráló lehet a tanuló számára, mert elindítja a tapasztalat okára vonatkozó kutatást. Véletlenül, élet adta esetekben az ilyen, Try kezdetű LAU-ok eredménye szélsőséges lehet. (Például az úszás tanulásának első lépéseként mély vízbe esés életveszélyes.) A gyors siker és a további tanulástól való merev elzárkózás egyaránt bekövetkezhet. A veszélyessége miatt a pedagógiai kihasználása, tervezése ennek a tanulási módnak nagyon megfontolandó.

Valójában nagyon ritka, hogy teljesen felkészületlenül találkozzon a tanuló egy-egy ismerettel, tudáselemmel. A tanulási folyamatba beépülő élmények tipikusan a korábbi ismeretek használatával kapcsolatosak. Optimális esetben a korábbi LAU-ok Create szintjéhez köthetők. Ebben az esetben az élményt a kapcsolatok felismerése, „összeáll a kép” öröme jelenti. A tanulási folyamat tervezése során fontos, hogy minél több ilyen élményhez jusson a tanuló.

A kutatás során megfigyeltem, hogy az élmények biztosítása alapvető szempont az oktatók, tanárok körében, azonban számos esetben rossz a kivitelezés. Az oktatóra jellemző, hogy szereti azt, amit tanít, aminek egyik legfontosabb oka, hogy sok pozitív élmény kapcsolódik a tananyaghoz. Ezeket az élményeket szeretné átadni a tanulóknak. Tipikus, hogy elmeséli a tapasztalatát, elmeséli, hogy az élete során hol, hogyan élte meg egy ismeret hasznosságát.

Példák oktatói élményre:

- „A karórám modellezése állapotgépként”. (Ki látja, milyen karóráról van szó?)
- „Három napig kerestem, hogy hol van hiba a kódban”. (Képzeljük el, milyen hosszú volt a kód.)

Az oktatás során nagyon fontosak az élménybeszámolók, de sok-sok óra hospitálása során meg kellett állapítanom magamra vonatkozóan is, hogy míg egy élmény elmesélése a mesélő

számára az élmény újra átélését jelenti, addig a hallgatója számára többnyire csak érdekes ismeret.[35] Pont a lényeget, a felfedezést, a problémák leküzdésének sikerét nem tudja az oktató átadni. Nagyon mély tartalma van ebből a szempontból az ismert idézetnek:

> *„A középszerű tanár magyaráz. A jó tanár indokol. A kiváló tanár demonstrál. A nagyszerű tanár inspirál."*
>
> *„The mediocre teacher tells. The good teacher explains. The superior teacher demonstrates. The great teacher inspires."*
>
> William Arthur Ward[36]

Az idézet felhívja a figyelmet arra, hogy a tanítás során a tanuló saját tapasztalata, saját élménye az, ami a tanulást, a fejlődést szolgálja és nem az oktató által átélt élmény.

3.2.1

IV/2. Gondolkodási képesség

A LAU folyamatot jellemez a fázisaival. A használhatóságának feltétele, hogy a tanulónak megvan a képessége a fázisok között szükséges átmenethez. A tanulás-tanítás tervezési folyamata, a módszerek és a kivitelezés is teljesen megváltozhat, ha egy képesség hiányzik, például siket a tanuló. Ugyancsak másképp kell tervezni, ha autista tanulóról van szó. Kevésbé látványos, de a LAU-ban leírt folyamatra hatással van, hogy a tanulónak milyen gondolkodási képességei vannak. Mennyire tud például memorizálni, analizálni, szintetizálni, strukturálni, szerializálni, felismerni mintázatokat.[37] Ahogy könnyen kimondható, hogy egy siket sohasem fog hallani, úgy könnyen kimondjuk, hogy a tanuló egy adott gondolkodási funkcióra képtelen. Azonban ennek (a siketség egyes eseteivel ellentétben) nincs tudományos alapja. A gondolkodási készségek fejleszthetők, azonban ezt nem lehet LAU alapján szervezni. A gondolkodás tanításának és tanulásának a specialitása, hogy a gondolkodás és az emlékezés nagyrészt történés, ami csak indirekt úton befolyásolható. Egy fogyókúrázónak azt „tanítani", hogy „ne gondoljon a csokoládéra", hogy „gondoljon arra, hogy nem fogja bírni a szíve" …, nem lehet. Pontosan tudja, hogy ezek jó tanácsok, jó tudások, de minél inkább nem akar gondolni a csokoládéra, annál inkább gondol rá.

[35] Vallomás: E dolgozat írása során is, újra meg újra átéltem az élményeket. A sikert, a kudarcot, a félelmet, az örömöt, a fájdalmat, a dühöt, a szeretetet, a bizalmat… A példákkal azokat az élményeket próbálom közvetíteni, amik engem inspiráltak. Remélem, hogy érdekesebbek ezek a példák a száraz magyarázatnál, de tudom, hogy kevésbé hatásos, mint a megélt eset.

[36] Forrás: https://www.brainyquote.com/quotes/william_arthur_ward_103463 [2021.10.30]

[37] Példák: Kisgyermekek felismerik a kapcsolatot az élő kutya, csiga, madár és a képeskönyv sematizált ábrája között. Megtanulja használni a kezét. Mindent szájába vesz – analizál. Mozgásában utánozza a környezetét.

Az agyban, idegrendszerben végbemenő folyamatok számunkra történések, amelyet indirekt módon, a környezet megfelelő alakításával tudunk befolyásolni. Azaz, ha nem akarjuk, hogy valami megtörténjen, akkor csökkenteni kell az esemény valószínűségét, másféle feladatokkal le kell kötni az agyat. Ha egy gondolkodási művelet véghezvitelét szeretnénk elérni egy tanulónál, akkor a környezetét ennek megfelelően kell alakítani és várni kell…, majd siker esetén megerősíteni, jutalmazni.

A hagyományos oktatásra jellemző, hogy a memorizálást, az ismeretek hatékony tárolását és visszaolvasását várja el, sokaknak problémát jelent az ismeretek feldolgozása. Az informatikaoktatás legfontosabb feladata az informatikai (számítógépes) gondolkodás „tanítása". Pontosabban: fejlesztése, hiszen ez egy képesség, aminek meglétéről már csecsemőknél meggyőződhetünk.

Általánosságban elmondhatjuk, hogy egy LAU kezdeti fázisaiban a befogadás (memorizálás) kap nagyobb szerepet, míg a tanulási egység vége az alkotáson (például informatikai gondolkodáson) alapul. Ezért is hiba a tanulási folyamatot a LAU^L3 szintig tervezni [77].

A LAU^L4-5 szintjén az alkotó gondolkodási tevékenységeknek kellene dominálnia a befogadóhoz képest. Azaz itt kap szerepet az informatikai gondolkodás, a kreativitás, a problémamegoldási képességek fejlesztése. Hospitáláson és a tanítási gyakorlatban is számtalan esetben találkoztam olyan tanulókkal, akikről az a vélemény, hogy „nem tudnak kreatívan gondolkodni",. nem tudják „kitalálni a megoldást". A vizsgált esetekben kiderült, hogy ezen vélekedés kialakulásában az IDŐ a legfőbb tényező. Egyrészt meggyőződés, hogy először kell az ismeretet befogadni, időben később felhasználni. Emiatt a befogadás biztosabban része a tanulásnak, mint az alkotás. (LAU^L1-3 valószínűbb, mint ugyanannak az ismeretnek a LAU^L5 fázisa). Másrészt a LAU^L1-3 szintek időben sokkal jobban tervezhetők. A tanulmányai során mindenkinek lesz egy tapasztalati képlete arra, hogy adott tananyagot mennyi idő alatt tud memorizálni. A problémamegoldás ideje ezzel szemben sokkal nehezebben tervezhető. Az „isteni szikrára" várni kell. Nehéz megmondani, mennyi idő alatt tudja valaki kigondolni a megoldást. Gyakran gyorsabb megkérdezni mástól, és megtanulni a választ. Ez azonban pont azt jelenti, hogy egy speciális esetre bemagolt megoldást kapunk a gondolkodási készség fejlődése helyett. A megfelelő tanítási módszerek kiválasztásánál ezért elsődleges volt annak figyelése, hogy a tanulás memorizálást (befogadást) vagy kreativitást (alkotást) igényel-e.

3.2.1

IV/3. Érzések, lelki állapot

A tanulás hatékonyságát jelentősen befolyásolják külső-belső egyéb tényezők is. A tanuló lelkiállapota, érzései az éppen tanult tananyaggal, illetve a külvilággal kapcsolatban. Ezek az érzések katalizátorként serkenthetik vagy gátolhatják a tanulást. Az oktatásmódszertanban gyakran előforduló kulcsszó a motiváció, amelyet különböző módszerekkel a tananyag elsajátítása irányába próbálunk fenntartani. Ilyen módszertani eszköz a napjainkban divatos gamifikáció is és az élmények biztosítása is. Emellett a mindennapi oktatási gyakorlatban figyelembe kell venni számos más tényezőt is. A tanításra felkészülés során lehet a tanulók lelki állapotára vonatkozó előképünk, de rugalmasan kell tudni alkalmazkodni a pillanatnyi helyzethez. Például: tudható, hogy ha az órát követően egy másik tárgyból dolgozat lesz, az jelentősen csökkenti a tanóra témájára való koncentrálás minőségét. A másik tárgy sikeres teljesítése jobban foglalkoztatja a tanulót. Hasonlóan, rontja a tanítás hatékonyságát a meleg, az éhség stb. De ennél kevésbé látható, a motivációt szintén befolyásoló tényezők is szerepet kapnak: fél társai véleményétől, összeveszett a barátjával szünetben, hiábavalók voltak a tanulásba fektetett energiái, környezete sikertelennek könyvelte el. Ismert tény, hogy a tanulást elsősorban a belső motiváció befolyásolja, a külső motiváció (tanári, szülői, társadalmi nyomás) hatása kisebb és sokszor ellenkezést vált ki.

A tanár lelki állapota éppúgy befolyásolja a sikert. Nagyon nehéz a termen kívül hagyni a külső problémákat. Elvárás, hogy a tanár a tanulóval szemben korrekt legyen, ne szimpátia alapon segítsen, értékeljen. Úgy legyen érzékeny a diákjai érzéseire, hogy közben kontroll alatt tartja saját érzéseit. A tanítás pedagógiai minőségét nagymértékben meghatározza, hogy a tanár azt várja-e el, hogy a diákjai alkalmazkodjanak az ő lelki állapotához vagy saját érzéseit félretéve, a diákok lelkiállapotához alkalmazkodva szervezi a tevékenységet.

Ahhoz, hogy a tanulás megvalósuljon, a tanuló kognitív tevékenysége szükséges. A szakmai és oktatási módszerek, „jó gyakorlatok” alkalmazása sokkal hatékonyabb, ha olyan érzelmi környezet társul hozzá, ami katalizátorként segíti a tanulási folyamatot. Kutatásom során többször találkoztam olyan helyzetekkel, ahol nem a megfelelő informatikai módszer és nem a megfelelő informatikaoktatási módszer kiválasztása jelentették a megoldást, hanem a módszerek alkalmazásához szükséges attitűd kialakítása.

3.2.1

IV/4. Tanulásmódszertan

A szaktárgy oktatásának módszertana alapvetően azzal foglalkozik, hogy az adott tananyagot az oktató hogyan adja át. Ezzel szemben a tanulásmódszertan a tanulást, a tananyag vételét

állítja középpontba. Sokszor elhangzik, hogy „először tanulj meg tanulni”, de az oktatásmódszertani kutatásokban ez általánosan nem jelenik meg. Vagy azért, mert adottnak tekintik, azaz a kutatás során egy adott tanulási módszert vizsgálnak, vagy azért, mert a kutatás a tanári tevékenység függvényében vizsgálja az eredményességet.

Az informatika tantárgyról sokszor mondják, hogy gyakorlati tárgy. Az a tapasztalatom, hogy ez a megjelölés – az elméleti tárgyakkal szembeállításként – azt jelenti, hogy nem oktató- vagy előadóközpontú, nem a befogadáson van a hangsúly, hanem tanuló-, azaz gyakorlatközpontú, az aktivitások vannak középpontban. Ideértve az informatikai gondolkodást is, ami kreatív, alkotó, gyakorlatias gondolkodás. Ezért kutatásomban fontos szerepet kap a tanulásmódszertan. Úgy látom, hogy az oktató mentor, tutor vagy tréner szerepek fontosabbak az előadói szerephez képest.

A tanulásmódszertan esetén sem tudom néven nevezni, hogy kitől, mit tanultam. Az interneten megtalálható tanulásmódszertan [42] tankönyv (ha akkoriban létezett volna,) lehetett volna az az alap, amire a gyakorlat során szerzett tapasztalataim rakódtak. Kulcsfontosságú a tananyag mellett a kompetenciák fejlesztésének vizsgálata; az élethosszig tartó tanulásban gondolkodás; a ma még ismeretlen, jövőbeni problémákra való felkészülés. A tanulás szervezése során cél a formális és informális tanulás egymást erősítő hatásának elősegítése; az önálló és vezetett, egyéni és kooperatív tanulási fázisok hatékony kialakítása; a motiváló munkakörnyezet, a flow-élmény [9] és a pihenés biztosítása.

3.2.1

V. Mi az Informatika?

A számítástechnikát a matematikatudomány olyan részének tekintették, amely egyre inkább elkülönül. George E. Forsythe, az informatikát önálló tudományként megjelenítők egyik úttörője 1968-ban így határozta meg az számítástechnika tudományt „What to Do Till the Computer Scientist Comes”[45] című cikkében:

> *”The most valuable acquisitions in a scientific or technical education are the general-purpose mental tools which remain serviceable for lifetime. I rate natural language and mathematics as the most important of these tools, and computer science as a third.”*
>
> *(nyers fordítás) „A természettudományos oktatás során szerezhető legértékesebb ismeretek azon az általános célú szellemi eszközök, amelyek egész életen át hasznosíthatók. A természetes nyelvek és a matematika ezen eszközök közül a legfontosabbak, a számítástechnika a harmadik.”*

Donald Knuth a fenti idézetre hivatkozva 1974-ben, „Computer Science and Its Relation to Mathematics” [46] című cikkében így egészíti ki a számítástechnika tudomány jövőjéről alkotott képet.

> *“Like mathematics, computer science will be a subject which is considered basic to a general education. Like mathematics and other sciences, computer science will continue to be vaguely divided into two areas, which might be called "theoretical" and "applied". Like mathematics, computer science will be somewhat different from the other sciences, in that it deals with man-made laws which can be proved, instead of natural laws which are never known with certainty. …”*
>
> *(nyers fordítás) „A matematikához hasonlóan a számítástechnika olyan tárgy lesz, amely az általános oktatás szempontjából alapvető fontosságú. Ahogy a matematika és egyéb tudományok, úgy a számítástechnika is nagyjából két részre osztható, amelyeket „elméleti”-nek és „alkalmazott”-nak nevezhetünk. A matematikához hasonlóan, a számítástechnika némiképp eltér a többi tudománytól, mivel az olyan ember által megalkotott törvényekkel foglalkozik, amelyek bizonyíthatók, a természetről alkotott tapasztalati törvények helyett. …”*

Kicsivel később a matematika és számítástechnika közötti különbségről:

"…mathematics dealing more or less with theorems, infinite processes, static relationships, and computer science dealing more or less with algorithms, finitary constructions, dynamic relationships."

(nyers fordítás) „…a matematika többé-kevésbé elméleti megfontolásokkal, végtelen folyamatokkal, statikus kapcsolatokkal; számítástechnika többé-kevésbé az algoritmusokkal, véges szerkezetekkel és dinamikus kapcsolatokkal foglalkozik."

Knuth cikkében olvashatjuk, hogy akkoriban egyes nyelveken másképp nevezik a Computer Science diszciplínát: Informatik, Datalogik, Data Processing. Pontosabban, akkor sem volt egyértelmű, hogy ugyanazt vagy mást, vagy bővebb diszciplínákat neveznek meg a fenti neveken.

2003-ban HE Shaoyi Informatics: the brief survey [47] cikkében írta le, hogyan értelmezi az informatika tudományt. Ennek magyar interpretációja Papp Lászlótól „Az informatika fogalma" [48], amiben az alábbi megállapítások olvashatók:

Mint egy fiatal és fejlődő tudományág egyre inkább magára vonja az információs szakemberek és kutatók figyelmét.

Az informatika, mint diszciplína meghatározásában kétértelműség figyelhető meg.

Saját alapvető fogalmainak kifejlesztésével egyre inkább úgy határozzák meg, mint különféle tudományterületek ötvözetét. Az informatika interdiszciplináris területként bontakozik ki, amely az információ és a technológia természetét tanulmányozza arra összpontosítva, hogy az emberek hogyan hozzák össze e kettőt annak érdekében, hogy előállítsák és menedzseljék (kezeljék) az információt és a tudást.

Az informatika alapvető aspektusa a tudást meghatározni, megszerezni, megosztani és hasznosítani a való világ különböző szervezeteiben…. Végül az informatikának foglalkoznia kell azzal is, hogyan lehet ezen a területen alkalmazni az értékelés eszközeit és módszereit.

Álljon itt a Wikipédián szereplő meghatározás [49] is, mely 2015-ös állapotot mutat. A szócikk forrásai közoktatási és szakképzési tananyagok, illetve szótárak:

Az informatika önálló tudományág, amely a különböző eszközökkel – de különösen a számítógéppel – megvalósított információkezeléssel, azaz az információ megszerzésével, (gyűjtésével), feldolgozásával, tárolásával, sokszorosításával és továbbításával foglalkozik.

V/1. Matematika vs. informatika

Természetes szám a nulla?

Az ELTE IK-n és a BME VIK-en oktatott programozás nagyon jellemző különbsége a listák indexelése. Az ELTE-n a listaelemek számozását a feladatok megfogalmazásából származtatva, általában 1-től kezdik, az indexelés i = 1-től i ≤ elemszám-ig tart, míg a BME-n az i = 0-tól i < elemszám használata vagy lezáróelem használata szokásos. (Ezt láthatjuk a kereséssel kapcsolatos példákban is.) Érvek mindkét használathoz bőségesen sorakoznak, kezdő programozótól a professzorokig mindenki preferál valamilyen megoldást. Dijkstra írása [50] 1982-ben született, véleményének súlyát személyes elismertsége adja. Az írással kapcsolatban sokan kifejtették véleményüket, ezek között egy, a Dijkstra említett írása kapcsán készült Hacker News-blog[38]. Érvek és ellenérvek, melyeknél a legfőbb szempont egy-egy, az értelmezéshez kapcsolódó másik fogalom vagy alkalmazás. Az épületek emeleteinek, illetve szintjeinek számozása, a születésnapok számozása, a kezdet létezése, a vég-kezdet értelmezése. (Pl. 10–14 óráig, ha minden egész órában be kell vennem egy gyógyszert, akkor hány gyógyszert kell reggel bekészítenem.) Dijkstra a természetes számok halmazának definíciójához is kapcsolja az érvelését:

> *„And the moral of the story is that we had better regard —after all those centuries! — zero as a most natural number."*
>
> *„És a történet erkölcsi oldala, hogy – sok száz év után! – a nullát természetes számnak kell tekintenünk".*

Az, hogy a nulla természetes szám-e vagy sem, a matematika területéhez kapcsolódó kérdés. Idézem a Wikipédián megtalálható véleményt erről:

> *„Mivel ez nem szorosabb értelemben véve matematikai probléma (nem lehet matematikai tételekből kiszámítani vagy bebizonyítani, természetes szám-e a nulla), hanem pusztán egy elnevezés tartalmáról való döntés, így definíció, megállapodás kérdése, hogy mi tartozik a névvel jelölt csoporthoz. A kérdés mégsem érdektelen, mert, bár a probléma nem matematikai jellegű, eldöntésének már vannak ilyen következményei – a feladatok, állítások, tételek rendszeresen hivatkoznak a természetes számok halmazára, és a feladat megoldhatóságát, a tétel érvényességét vagy bizonyíthatóságát döntheti el a fogalom értelmezése."*

[38] Hacker News (2011.06.21) https://news.ycombinator.com/item?id=2679769 [2021.10.30]

Pusztán egy elnevezés tartalmáról szóló döntés, ami nem matematikai jellegű probléma? Akkor milyen probléma? Filozófiai probléma vagy esetleg erkölcsi? A döntések gyakorlata informatikai probléma, ezért ez egy (erkölcsi) visszahatása az informatikának a matematikára?

Én még úgy tanultam az általános iskolában (197x) matematikából, hogy a nulla nem természetes szám, de az egyetemen (198x) már azt tanultam, hogy természetes szám. Egy 2017-ben írt szakdolgozatban [51] az első Peano axióma a $0 \in \mathbb{N}$, de lábjegyzetben megjegyzésként olvashatjuk, hogy eredetileg 1 volt a 0 helyett, valójában megállapodás kérdése, a szerző a közoktatásban elfogadott értelmezést használja. A programtervező informatikus képzésben, a tervezési szakaszban, amelyben jellemzőbb a matematika hatása, mégis inkább az 1 a sorozatok első indexe, ami megfelel a közoktatásbéli – és ezért közismeretnek megfelelő – sorozatdefiníciónak. A közoktatás mentén tovább vizsgálva a témakört, az egyik legismertebb feladat:

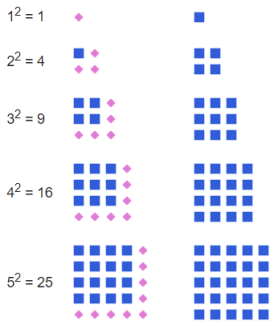
Példa:

Adjuk meg az n. négyzetszámot, majd ezt követően adjuk meg az első n négyzetszám összegét.

Érdemes megvizsgálni, mit ír erről a Wikipédia[39] (mivel sok diák ezt nézi meg).

„A számelméletben négyzetszámon vagy teljes négyzeten (teljes második hatványon) olyan egész számot értenek, amely felírható valamely egész szám négyzeteként…”

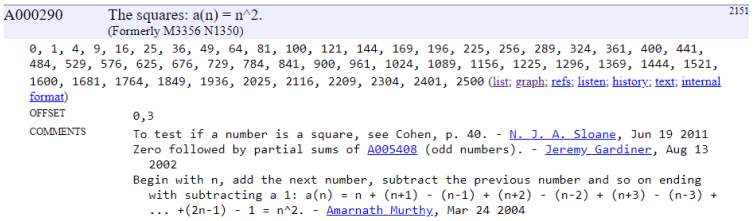
Ezt követően látható a 26. ábra



26. ábra: Négyzetszámok ábrázolása

Majd: *„$0^2 = 0$, a nulladik négyzetszám”* megállapítást követően „Példa”-ként egy lista olvasható a négyzetszámokról, hivatkozva a https://oeis.org/A000290 oldalra, ahol – ahogy a 27. ábra mutatja – az első érték a 0.

[39] Wikipédia: Négyzetszámok (2018.11.27) https://hu.wikipedia.org/wiki/N%C3%A9gyzetsz%C3%A1mok [2021.10.30]



A000290 The squares: a(n) = n^2. 2151
(Formerly M3356 N1350)

0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400, 441, 484, 529, 576, 625, 676, 729, 784, 841, 900, 961, 1024, 1089, 1156, 1225, 1296, 1369, 1444, 1521, 1600, 1681, 1764, 1849, 1936, 2025, 2116, 2209, 2304, 2401, 2500 (list; graph; refs; listen; history; text; internal format)

OFFSET 0,3

COMMENTS To test if a number is a square, see Cohen, p. 40. - N. J. A. Sloane, Jun 19 2011
Zero followed by partial sums of A005408 (odd numbers). - Jeremy Gardiner, Aug 13 2002
Begin with n, add the next number, subtract the previous number and so on ending with subtracting a 1: a(n) = n + (n+1) - (n-1) + (n+2) - (n-2) + (n+3) - (n-3) + ... +(2n-1) - 1 = n^2. - Amarnath Murthy, Mar 24 2004

27. ábra: Négyzetszámok felsorolása

Majd ezt követi az első tulajdonság: *Az n-edik négyzetszám képlete n²…:*

$$n^2 = \sum_{k=1}^{n} (2k - 1)$$

Melyik az első négyzetszám? n = 0 esetén hogyan értelmezhető a képlet? Az a megállapítás, hogy a „0 a nulladik négyzetszám", zavart okoz, mivel a sorozatok indexelése 1-től kezdődik és semmi nem utal arra, hogy átértelmezték volna a sorozat definícióját. A négyzetszámok nagyságszerinti sorozatában: 0, 1, 4, 9…. – az első a 0. A képlet nem zárja ki az n = 0 esetet, de nem is mond rá értelmes számítási módot. Kétségtelen, hogy kevésbé szép, de kivédi ezt az alábbi képlet az *n*. négyzetszám értelmezésére és számítására:

$$(n-1)^2 = \sum_{k=0}^{n-1} (2k - 1) + 1$$

Szabadon eldönthető, hogy a nulla természetes szám-e. Ehhez képest szabadon eldönthető, hogy a sorozatok értelmezési tartománya a természetes számok halmaza vagy a pozitív egészek halmaza. Az eredmény a magyar közoktatásban egy matematikailag értelmetlen definíció, amely szerint annak a sorozatnak, amelyiknek definíció szerint nincs nulladik eleme, mégiscsak van nulladik eleme, de ez nem eleme a sorozatnak.

A probléma nem egyedi, például a Fibonacci-sorozatnál (első két elem 0, 1 vagy 1, 1 és ezeknek mi az indexe), az 5. elem értékének meghatározása, az *n*. páros szám értéke pusztán az értelmezési pontatlanságok miatt gondot okoz, mert a természetes számok halmazának, illetve a sorozatnak a definíciója, esetlegesen tartalmazza a 0-t. A probléma akár matematikai, akár nem, a vegyes döntésnek gyakorlati következményei vannak, ezért nem filozófiai probléma. Adott definíciók (megnevezések) értelmezése alternatív, amelynek következtében módosulnak a tételek. Az értelmezésből adódó modellezés, azaz a specifikáció a kulcs. Az adathoz más értelmezést rendelve módosul az információ, másik modellt kapunk, ezért – azt gondolom – az informatika tudományhoz tartozó problémáról van szó. De ha a döntés az informatika tudományához tartozik, akkor miért vitatkoznak rajta évtizedeken át az informatikusok? El fog jönni az az idő,

amikor egyértelműen 0-tól indexelnek minden sorozatot? Esetleg lesz egy forradalom, amikor eldöntik, hogy 1-től kell indexelni a sorozatokat? Lehet-e a Dijkstra által adott „erkölcsi" támogatás eredménye az, hogy 100%-os dominanciája legyen az egyik értelmezésnek? Azt gondolom, hogy az informatikának nem ez a feladata. Az informatikus mindig az adott specifikációhoz fogja igazítani az értelmezést, ha hiányzik a pontos értelmezés, akkor minden lehetséges értelmezést figyelembe kell vennie. Ha a matematikában véglegesen úgy építik fel a tételeket, hogy a 0 természetes szám és a sorozatok értelmezési tartománya a természetes számok halmaza lesz… az informatika akkor is az aktuális feladat és a programozó számára legkifejezőbb (humán kategória) módon fogja indexelni a sorozatokat, listákat. A matematikai definíciótól, axiómától függetlenül értelmezik az informatikai modellek a 0 szerepét, mindkét értelmezést gyakorlati helyzetekben alkalmazva.

Függvény

A függvény fogalmának eltérő értelmezését saját bőrömön tapasztaltam. A feladat így szólt[40]:

> *„ …Milyen okból korlátozza a double (*f)(double) paraméter a megvalósítást? Találj ki egy módszert, hogyan kerülöd meg a problémát, és valósítsd is meg C-ben! Gondolj arra, hogy igazából a sin(d*x) csak egy példa… Lehetne d*sin(x), d*sin(c+x), esetleg a*sin(b*x+c)+sqrt(d*x*x) is a függvény, amit ábrázolni kell."*

A feladat lényege, hogy az *x* változón kívül paramétereket – egyet, kettőt, sokat – is kellene kezelnie a megírandó függvénynek. Azonban az „a * sin(b * x + c) + sqrt(d * x * x)" függvény ábrázolásának felvetése számomra egy matematikához készítendő programot jelentett, amihez a skalárral szorzást, függvények összegét és kompozícióját készítettem el. Számomra az általánosított feladat egy függvényábrázoló program volt, amihez még a kifejezés kiértékelését is megírtam. Egyhónapos munka után megkérdeztem, hogy elég jó lett-e. Némi értetlenkedés után, hogy miért is bonyolítottam el ennyire a feladatot, megértettem, hogy a különböző típusú függvények tetszőleges kompozíciója a matematikatanítás érdekes feladata, a függvényanalízis matematikus probléma. A gyakorlati alkalmazások során egyértelmű, hogy milyen függvény az, amit használni kell (legyen az hanggenerálás vagy röppályaszámítás), ennek a változó mellett csak a paraméterei átadását kell megoldani, a kiszámítási módot a konkrét esetre megírják (kódolják).

[40] C11 és C++11 programozás (választható tárgy) 1. labor 1. feladat, BME VIK 2017.

- Matematikában a függvény egy – jellemzően számok közötti – összerendelési szabály (egyértelmű reláció), aminek vizsgáljuk a szélsőértékeit, gyökeit, menetét, folytonosságát és számos tulajdonságát.
- Informatikában a függvény egy programkód, ami a bemenetén kapott argumentumokat is felhasználva – esetleg mellékhatást is okozva – van, hogy létrehoz a kimenetén egy eredményt.

Az informatika függvényfogalma a matematika függvényfogalmához képest nem csak általánosabb, de más a szerepe, mások a jellemzői. Ezzel együtt a matematika függvényfogalma – más értelemben véve – általánosabb. Matematikatanári szemmel a példában más tulajdonságok megadását véltem megtalálni, mint amire a feladat informatikus kitűzője gondolt.

4.1.1

VI. Informatika – az egység kétsége

VI/1. Példa a gondolkodásmódok különbségére

Tantárgyi jegyzetek összehasonlítása

Az ELTE-n programozás vizsgámon beugró kérdés volt valamelyik programozási tétel algoritmusa. Ma is „zsigerből” kell tudni az eldöntés, a keresés és a kiválasztás tételét, a hasonlóságot és a különbségeket is a specifikációk és az algoritmusaik között. Az alábbi részletek az ELTE Programozás tárgyához készült előadás jegyzet[41] illetve a BME *Programozás alapjai 2* tárgyhoz fejlesztett C++ jegyzetben[42] találhatók.

ELTE IK	BME VIK
Eldöntés	

Specifikáció:
Bemenet: $N \in \mathbb{N}, X \in \mathbb{H}^N, T: \mathbb{H} \to \mathbb{L}$
Kimenet: $Van \in \mathbb{L}$
Utófeltétel: $Van = \exists i\ (1 \le i \le N): T(X_i)$
Algoritmus:
i ≔ 1
i ≤ N és nem (T(X[i])
i ≔ i+1
Van ≔ i≤N
	Generikus implementáció:
	Tarolo_eleme_e
	template<typename ITER, typename T>
	bool Tarolo_eleme_e(ITER first, ITER last,
	T const& value) {
	for (ITER it = first; it != last; ++it)
	if (*it == value)
	return true;
	return false;
	}

Keresés	

Specifikáció:
Bemenet: $N \in \mathbb{N}, X \in \mathbb{H}^N, T: \mathbb{H} \to \mathbb{L}$
Kimenet: $Van \in \mathbb{L}, Ind \in \mathbb{N}, Ert \in \mathbb{H}$
Utófeltétel: $Van = \exists i\ (1 \le i \le N): T(X_i)$ és
$Van \to 1 \le Ind \le N$ és $T(X_{Ind})$ és $Ert = X_{Ind}$
Algoritmus:
i ≔ 1
i ≤ N és nem (T(X[i])
i ≔ i+1
Van ≔ i≤N
1 Van N
Ind ≔ i; Ert ≔ X[i]
	Generikus implementáció
	template<typename ITER, typename T>
	ITER Tarolo_keres(ITER first, ITER last,
	T const& value) {
	for (ITER it = first; it != last; ++it)
	if (*it == value)
	return it;
	return last;
	}

[41] Szlávi – Zsakó: Programozás 3. előadás. (ELTE IK. IP-18PROGEG) jegyzet diái: 37., 43., 50.; 2018. http://progalap.elte.hu/downloads/eloadas/progalap_ea3.zip [2021.10.30]

[42] Dobra Gábor: CPPFTW Prog2 segédlet 6. fejezet 2019. 01. 07. https://prog2.cppftw.org/ea06/#3 [2021.10.30]

Az ELTE-n elvárás a strukturált (módszeres) programozás. Az, hogy a fenti generikus függvényekben két return van, ennek az elvárásnak a megsértése, mert a programozási tételben megfogalmazott feltételes ciklus helyett a számlálós ciklus megszakítását láthatjuk. A strukturált programozásnak megfelelő kód az adott esetben ilyen lenne:

```
template<typename ITER, typename T>
ITER Tarolo_eldont(ITER first, ITER last, T const& value) {
  ITER it = first;
  while(it != last && *it != value)
    ++it;
  return it;
}
```

A szakma meghatározó képviselőinek véleménye

A strukturált programozás szabályaitól való eltérés rengeteg szakmai vita forrása. Az elmélet oldaláról Dijkstra 1968-ban támadja a vezérlés szabálytalan átadását [55]. Dijkstra levelének megjelenése után 40 évvel Robert C. Martin a tiszta (jól tervezett) kódról írt Clean Code… [56] könyvében 29-szer szerepel az alábbi kódminta:

```
fPrefix = 0;
for (; fPrefix < end; fPrefix++) {
  if (valamifgv(fPrefix))
    break;
}
```

Emellett 17-szer található meg a while(/*…*/), még kétszer while (true) valamint kétszer do {} while (/**/) ciklusszervezés. Azaz a strukturált programozás egyik alapelvét az esetek 58%-ában sérti a tiszta, szép kódról szóló könyv.

Elméleti és gyakorlati megfontolások

A lineáris keresés algoritmusa és implementációja között további eltéréseket is megfigyelhetünk. A keresés tétel (ahogy az ELTE-n tanítják) a ciklus után egy elágazást is tartalmaz, ami az implementációs példánkban nem szerepel; a függvény hívójának kell ellenőriznie, hogy van-e találat. A last értékkel visszatérés nyelvi és implementációs specialitás. A strukturált programozás elvei alapján egy keresés eredménye a megtalált elem kérdezett adata, vagy az, hogy nincs meg az elem.

A Tarolo_keres függvény a Tarolo_eleme_e eldöntéssel szinte azonos, feleslegessé teszi az eldöntés függvény megírását azáltal, hogy az eredmény kiértékelését a hívóra hagyja.

Az, hogy egy, mérnökinformatikus szempontból felesleges, Tarolo_eleme_e függvény az idézett jegyzetben megjelent, lehet a kutatásom mellékhatása, mivel BME-s oktatóval beszélgetve mondtam, hogy a két feladat megkülönböztetése elvárás az ELTE-n. Kezdetben a C nyelvben nem volt logikai típus, így az összes logikai függvény egész számmal tér vissza, melyet a „0 azonos a hamis-sal, minden más érték igaz” szabállyal alkalmaztak logikai értékként. A C nyelv

is fejlődik, 2017-ben „nagykorú lett” az stdbool, amely tartalmazza a bool adattípust a C nyelvi fordítók számára. A változás a jegyzetekben nyomon követhető. A *Programozás alapjai 1*. tantárgy 2014 őszi jegyzetében az eldöntés tétel megjelenése egyetlen címben szerepelt, a keresés tétellel egybevonva: *7. előadás Tömbök algoritmusai 5. pont: „Van-e?” – lineáris keresés/eldöntés*. A 2018-as jegyzetben már mindenhol megkülönböztetve szerepel a keresés és az eldöntés. Az eldöntés tétele külön pontként jelenik meg a 2. előadás *Nevezetes algoritmusok, tömbök*, a 3. előadás *Függvények* és a 6. előadás *Tömbök algoritmusai* fejezetben. Azonban a két algoritmustípus között csak a visszatérési érték típusában van eltérés, ahogy ezt az idézett jegyzetben is láthatjuk.

A keresési és eldöntési probléma megkülönböztetése programtervezéskor elvi kérdés, mérnökinformatikusként inkább esztétikai kérdés. Strukturált programozással megoldva a feladatot, a `Tarolo_keres` függvényben a keresés tételét implementálva, duplán kellene elemezni az eredményt: először a függvényen belül lenne egy elágazás majd a felhasználáskor. További megvalósíthatósági nehézséget okoz, hogy ha nincs találat, akkor a visszatérési érték típusa más lenne. Ezért a keresés eredményének függvényen belüli kiértékelése 1. hibás lehet, ha nincs találat; 2. csúnya lehet, ha a hiba típusát az eredmény típusával azonossá teszi; 3. eljárásként megírva egy összetett problémán belüli felhasználása nehézkes.

A két gondolkodásmód a keresési problémákat eltérő végeredménnyel oldja meg. A programtervező az eredményt kiértékelve adja meg a választ, míg a mérnökinformatikus a kiértékelést a további felhasználóra bízza. Mindkét esetben módszertanilag (szakmailag) alátámasztott megoldásról van szó, azaz nem képző intézményi specifikum a jelenség.

Eltérő preferenciák hatása az oktatásra, hallgatók gondolkodásának formálása

Tapasztalatom az is, hogy az ELTE-n az elsőéves hallgatók a keresés ciklusát nem írják külön függvénybe, összevonják azzal amire a keresés eredményét fel akarják használni. Így a kiértékelést ők is csak egyszer végzik el. Kezdő programozás tárgyak tekintetében, a tárgy céljaiból adódóan az is jellemző, hogy az ELTE-n a feltétel általában összetettebb, mint a BME-n. Az ELTE-n a szemeszter végén jellemző, hogy érdemes lehet a keresés feltételének külön függvényt írni (specifikációban „T tulajdonság”, `bool T(X[i])`). A BME-n gyakoribb, hogy a keresés a listában található konkrét értékre vonatkozik, a beszúrást és a törlést előkészítve; a bonyolultabb feltételek implementálása a ciklusmagban a természetes gondolkodásmódot követi, nincs motiváció a feltétel külön függvényben történő megvalósítására. Az eltérő megközelítésből adódik, hogy a programtervezőknél az algoritmuson, míg mérnökinformatikusoknál az adat implementációján van a hangsúly. Az eltérő megközelítés eltérő dekompozíciót eredményez.

A BME-n az eldöntési és a keresési feladatok közötti különbség elenyésző, de a kiválasztás sem különül el. Nem foglalkoznak azzal, hogy a specifikáció alapján biztosan van-e találat, keresésként fogalmazzák meg a kiválasztás tétellel megoldható feladatokat is. Az indok: „Soha nem lehet biztosan tudni, hogyan, mire akarom még a kódrészletet használni. Mikor lesz végzetes következménye annak, hogy biztos találatot feltételeztem.” Azonban van egy másik elv is: „Biztosítom, hogy legyen találat.” Ennek a gondolkodásmódnak a következménye a „strázsa” (sentinel), a lista végén levő lezáró elem vagy a lista végét jelentő null.

Kiválasztás

Specifikáció:

Bemenet: $N \in \mathbb{N}, X \in \mathbb{H}^N, T:\mathbb{H}\to\mathbb{L}$
Kimenet: $Ind \in \mathbb{N}, Ert \in \mathbb{H}$
Előfeltétel: $N\geq 0$ és $\exists i\ (1\leq i\leq N):T(X_i)$
Utófeltétel: $1\leq Ind\leq N$ és $T(X_{Ind})$ és $Ert = X_{Ind}$

Algoritmus:

```
i := 1
nem (T(X[i])
    i := i+1
Ind := i; Ert := X[i]
```

A programtervező megkülönbözteti az „elemeken végig megyek” és az „elemeken addig megyek amíg” algoritmust. A mérnöki gyakorlatban ez a két megfontolás egybeolvad, a „végig” helyett az „elemeken addig megyek, amíg vannak” megfogalmazás használatával. A pointer a kötelezően NULL értékű, elemek után található last eléréséig változik. A „vége” az utolsó utáni elem, a biztosan létező NULL. A programtervező azt mondja: „A feladat alapján egyszerűsíthetem az algoritmust, a keresés helyett elég kiválasztani, nem kell figyelni a lista végére”. A mérnökinformatikus viszont így gondolkodik: „Mindig probléma a keresés során, ha nincs találat. Gondoskodom arról, hogy legyen találat. Majd a felhasználó eldönti, hogy valódi-e a találat.”

4.2
6.2.4

VI/2. Egy feladat algoritmikus és OOP megoldása

A programtervező informatikus „elméleti” tervező szemléletmódja és a mérnökinformatikus megvalósításra fókuszáló programozási „gyakorlata” más paradigmák mentén értelmezi és oldja meg a problémákat. Az Eratosztenészi szita implementációin jól érzékelhető a programtervező algoritmikus és a mérnök objektumra, adatra fókuszáló gondolkodásmódjából adódó különbség:

Példa: Eratosztenészi szita.

Adjuk meg egy adott *N* értékig a prímszámokat! [43]

[43] OOP absztrakció forrása: C11 és C++11 programozás (VIEEAV01) tantárgy, 2. labor 1. feladat (2017/2018 II. félév, évente megújuló, belső anyag) https://cpp11.eet.bme.hu/lab02/index.html [2018.04.14]

Strukturált, procedurális megoldás

Vegyünk fel egy N + 1 elemű logikai tömböt, melynek 0. és 1. eleme false, a többi kezdetben true.

- Egy iterátorral menjünk végig a tömb elemein N gyökéig és minden esetben, amikor a mutatott elem true,
  - az adott helyindex többszöröseinek minden létező helyén levő értékét állítsuk false-ra.
- A prímszámok listáját a tömb true értéket tartalmazó indexei adják.

Objektum orientált megoldás

Készítsünk egy két végén strázsás, kétirányban láncolt listát, amelynek

- kezdete egy adagoló (a kapott számot tovább küldi);
- a vége a prímszámok kiírója, azaz gyűjtője, a vödör (a kapott számot kiírja és beszúr maga elé egy adott értékű szűrőt;
- a közbenső elemek az eljárás során keletkező szűrők, mely a kapott számot csak akkor küldi tovább, ha a saját értékével a szám nem osztható.

Kód (C# nyelven, $N = 100$ értékkel)

```
using System;
namespace Eratosz_Proc
{
class Program
{
  static void Main()
  {
    bool[] prim = new bool[101];
    prim[0] = false;
    prim[1] = false;
    for (int i = 2; i <= 100; i++)
      prim[i] = true;
    int ix = 2;
    while (ix * ix <= 100)
    {
      if (prim[ix])
      {
        int m = 2;
        while (m * ix <= 100)
        {
          prim[m * ix] = false;
          m++;
        }
      }
      ix++;
    }
    for (int i = 0; i <= 100; ++i)
      if (prim[i])
        Console.Write(i + ", ");
  }
}
}
```

```
using System;
namespace Eratosz_OOP
{
  abstract class Elem {
    public Elem Prev { get; set; }
    public Elem Next { get; set; }
    public abstract void Todo(int szam);
  }
  class Adagolo : Elem {
    public override void Todo(int szam) {
      Next.Todo(szam);
    }
  }
  class Szuro : Elem {
    private readonly int ertek;
    public Szuro(int ertek) {
      this.ertek = ertek;
    }
    public override void Todo(int szam) {
      if(szam % ertek != 0) Next.Todo(szam);
    }
  }
  class Vodor : Elem {
    public override void Todo(int szam) {
      Console.Write("{0}, ", szam);
      Szuro uj = new Szuro(szam);
      uj.Prev = this.Prev; uj.Next = this;
      this.Prev = uj; uj.Prev.Next = uj;
    }
  }
  class Program{
    static void Main()   {
      Adagolo a = new Adagolo();
      Vodor v = new Vodor();
      a.Next = v; v.Prev = a;
      for (int i = 2; i <= 100; i++) a.Todo(i);
    }
  }
}
```

Míg a procedurális megoldásban egy sorozatszámítás és egy kigyűjtés kombinációját láthatjuk, az OOP megoldásban a szűrő objektumok továbbadják egymásnak a számukra nem megfelelő számot. A vödör objektumig jutó szám a kiírás mellett egy újabb szűrőt eredményez.

4.4
6.2.4

VI/3. Informatikai fogalmak értelmezése

Néhány, az informatika oktatása során használt (tanított, számonkért) fogalom, amelynek tartalma csak az adott környezet pontos ismeretében tisztázható:

- Adatmennyiség mértékegységének váltószáma, a kilo értelmezése (1000 vagy 1024).
- Digitális világ – ami körül vesz, vagy ami a gépben van?
- Dinamikus tömb – amit a program futása közben egyben lefoglalunk, illetve a dinamikusan foglalt azonos típusú adatokat tartalmazó sorozatok.
- MAC-address – a hálózati kártyától elválaszthatatlan(?) egyedi(?) fizikai azonosító.
- Mesterséges intelligencia narrow, general és super szintként jelölt fogalmai.
- Operációs rendszer definíciója PC-re, felhőben vagy virtuális gépen futó „gép” esetén.
- Patch-kábel fogalma: a rövid átkötő kábel, illetve a lengőkábel értelmezéssel
- Periféria – a processzorhoz képest, a kézben tartott eszközhöz képest
- ROM → PROM → EPROM → EEPROM. Az „electrically erasable programmable read-only memory” lényegében eredeti önmagának az ellentéte.
- Struct nyelvtől függően lehet record, tuple értelmű vagy class. Egyes definíciókban csak adattagot tartalmazhat, máskor függvényt is.

4.5

VI/4. Az Informatika esztétikája – szépség és tisztaság

A kutatás során sokszor külső, majd belső konfliktusaim voltak a „szép megoldás”, avagy a minőségi megoldással kapcsolatban. Csak néhány példa:

- Hol, hogyan jelenjenek meg a blokkok nyitó és csukó zárójelei?
- Lokális vagy globális változót preferáljak?
- Figyeljek-e az indexelő (iterátor) előjelességére?
- Hátul teszteljem a ciklust vagy lehet a logikai kifejezésnek mellékhatása? Jó-e, szép-e a „while(getline())”?
- Logikai kifejezést írjak vagy Boole-algebrát használva szorzást és összeadást? (Például: táblázatkezelésben az ÉS(A1;A2) függvény vagy az A1*A2 a szebb, a bemutatandó megoldás?)

– Mennyire bontsam a feladatot részeire, mely eljárásokra, kifejezésekre írjak függvényt?

Jellemző példa a pre- vagy postinkrementálás kérdése – i++ vagy ++i használata. Néhány tipikus érv és ellenérv:

– Amikor mindkettő használható, akkor a ++i egy másolással kevesebb műveletet jelent.
– Az i++ a másolás miatt biztonságosabb.
– Mindegy melyiket használjuk, mert a programozási nyelv fordító programja tipikus helyzetekben optimalizálja a kódot, az i++ is ++i lesz.
– Az i++ jobb, mert gyorsabban gépelhető, a ++i gyakran lesz ++I és a javítás lassít. A használt fejlesztő környezet automatikusan javítja a kis- és nagybetűs írást.

A kutatás során a külső konfliktust számomra az jelentette, hogy a sokéves gyakorlat során megszokott megoldásaimat a BME-n jó esetben nem láttam, rosszabb esetben megszólták. Miután megtanultam a helyi szokásokat, az iskolai feladatok megoldása okozott gondot: az adott helyzetben melyiket is kellene alkalmaznom, melyiket kellene tanítanom. Emiatt néha a legegyszerűbbnek tűnő feladat megoldása is jóval több időbe tellett, mint az elvárható lett volna, továbbá jellemző volt, hogy végül többféle megoldást készítettem.

Mitől szép?

A fenti példákhoz képest kevésbé általánosítható, de jellemző példa az Ötszáz nevű érettségi feladat[44] fizetendő összeget kiszámító függvénye, amelyet az Igy írtok Ti… – korábban már említett előadásunkon [59] Czirkos Zoltán elemzett.

Programtervezői szemmel, az érettségi feladat mintamegoldása jó példája a probléma matematikai megoldásának. Látva a megoldást, tetszett a kompaktsága. A központi megoldás matematikai tudásra és strukturált programozásra épít. Teljes if-else struktúra látható benne, bár a két visszatérési pont (két return) sérti a strukturált programozás elveit, ezzel megspórol egy változót:

```
if(db<=2) {return db * 500 – (db – 1) * 50;} /*db = 1: 1 * 500 – 0 * 50, db = 2: 2 * 500 – 1 * 50 */
else {return 1350 + (db – 3) * 400;} /* db = 3: 1350 + 0 * 400 */
```

Azonban ez a megoldás egy mérnökinformatikus számára csúnya: Egyrészt, a fenti megoldásban az „else”-nek nincs értelme a „return” után. A változóval való spórolás (egyszerűsítés) felesleges, mert ezt az optimalizálást a fordító is el tudja végezni, a számítást tároló lokális változóval legalább tisztán strukturált lenne a kód. Másrészt, a kiszámítási szabály nem tükrözi

[44] https://www.oktatas.hu/bin/content/dload/erettsegi/feladatok_2016tavasz_emelt/e_infmeg_16maj_ut.zip [2021.10.30]

a feladat szövegét, amely szerint az első termék 500 Ft, a második 450 Ft, a továbbiak 400 Ft-ba kerülnek. A kódrészlet a feladat szövege alapján így nézne ki:

```
if (db == 1) return 500;
if (db == 2) return 500 + 450;
return 500 + 450 + (db – 2) * 400;
```

A két konstans kiírásának szépsége vitatható, hiszen ennyit fejben megold az ember… Mégis, ez a biztosabban igaz, ez következik természetes módon a feladat szövegéből, és ezért ez a szép megoldás. Ráadásul műveletek számát tekintve sem rosszabb így, hiszen a gép számára a fordítóprogram fogja előállítani a bináris kódot, ami optimalizálás után a konstansok összegét fogja beírni, míg a szabályok módosítása esetén könnyű megkeresni a kódban módosítandó konstansokat.

„Költői” kérdések: Melyik megoldás szebb? Melyik jobb? Melyik megoldási módot tanítsuk? Melyik elveket alkalmazzuk? Mennyire bízzuk a kódunkat a fordítóprogram optimalizálási képességeire? Ha az egyik megoldási módot példaként bemutatjuk, akkor a másikat hogyan értékeljük? Mely érvek a fontosabbak? Mit kezdjünk a tökéletesen ellentmondó érvekkel?

4.6

VII. Az informatika tantárgyi megjelenése

VII/1. Az informatika helye a tantervekben

Magyarországon az 1980-as évektől a technika tantárgy keretében próbálták tanítani a számítástechnikát. A technika korábban gyakorlati tantárgy volt, fa- és fémmegmunkálás, kötés, varrás, villanyszerelés volt benne, külön fiúknak és lányoknak, külön vidéki és városi környezetre. A számítástechnika témán belül elsősorban a gép kezelését, az operációsrendszert és a programozás alapjait tanulták a gimnáziumokban [63]. A programozás a 80-as évek végére a matematikatantárgyban is választható tananyag volt [72], de csak néhány iskolában volt kísérlet a megvalósításra. Alkalmazásból 1990 körül a CHI Writer volt a szövegszerkesztő és Quattro a táblázatkezelő program. A tananyagot az iskolák, tanárok egyénileg határozták meg. Volt, ahol csak számítástechnikát tanítottak technika órakeretében, máshol fele-fele megoszlásban tanultak Szűcs Ervin: Technika könyveiből [71], illetve számítástechnikát gyakorlatként, vagy csak elméleti technikát tanítottak. Érdekes, hogy ekkor a gimnáziumi technika – bár műhelyt és tankonyhát is terveztek az ekkor épülő gimnáziumokba – elméleti volt. Lakás energiaellátása, vízrendszer, hőtárolás, modell, ember és környezete… ehhez hasonló fogalmakat tanultak a diákok. Egy körülbelül 1995-ös őszi javító-pótló központi érettségin kérdezőtanár voltam számítástechnika-technikából. Számítástechnikatanári végzettségemmel alkalmasnak gondoltak technika érettségiztetésre is. A megjelenő körülbelül 5 diák annyira különböző témákat tanult, hogy el kellett kérnem az eredeti iskolájukból a vizsgatematikát és a technika témához segítséget kellett kérnem. A sokszínűség akkoriban szervezési problémát jelentett, amely más tantárgyakra nem volt jellemző. A helyzeten jelentősen változtatott az 1995-ös NAT [64], amely megnevezte az Informatikát műveltségterületként, miközben a Technika tantárgy az Életvitel és gyakorlati ismeretek műveltségterületen belül újjászületett.

Az 1995-ös NAT-ban [64] az Informatika műveltségterület behatárolása:

> *„… Ehhez el kell sajátítania a megfelelő információszerzési, feldolgozási és -átadási technikákat, valamint az információkezelés jogi és etikai szabályait (információk átvétele, bizalmas kezelése stb.). E gyorsan változó, fejlődő területen nagyfokú az ismeretek elavulása is, ezért különösen fontos, hogy a tanulónak igénye legyen informatikai ismereteinek folyamatos megújítására.*
>
> *A felkészítés fő területei:*
>
> - *a számítógépes ismeretek,*
> - *a könyvtári informatika,*

- *az információkezelés technikai oldala (az információszerzés, -feldolgozás, -tárolás és -továbbítás fizikai megvalósításai – lásd a Technika fejezetben),*
- *a tömegkommunikáció (lásd a Művészetek Mozgóképkultúra és médiaismeret című fejezetében).”*

Az Életvitel és gyakorlat műveltségterületen belül a Technika lényege:

„A Technika a megvalósításra, a működésre koncentrál, elsősorban a tevékenységre épít. Nem az utánzásra, hanem az alkotásra, az önálló problémamegoldásra készít fel, és így jelentősen járul hozzá a rugalmas gondolkodásra neveléshez.” Témái között megtaláljuk a közlekedést, háztartástant, életvitelt, mezőgazdasági ismereteket.

Megfigyelhető, hogy az Informatika tantárgy az információval, az adattal foglalkozik, míg a Technika tantárgy az anyaggal. Ugyanakkor látszik a közös terület is, a modellalkotás, a gondolkodásfejlesztés, a problémamegoldás területén. Amennyiben a tantárgyak gyakorlati oldalát nézzük, az Informatika a számítógépes, a számítógéppel végezhető munkában való gyakorlatot biztosítja míg a Technika azon túl, az anyagmegmunkálás, a mindennapi élet eljárásaiban ad gyakorlati ismeretet.

A 2003-as [65] (2007-ben továbbfejlesztett [66]) NAT a kompetencia alapú oktatás tanterve. Jellemző, hogy míg az 1995-ös tanterv bevezetőjében 3-szor, a 2003-as NAT-bevezetőjében 63-szor fordul elő a „kompetencia” szó. Az előző tantervhez képest jellemző a spirális tervezés, azaz minden résztémára megadja a fejlesztési feladatokat az 1–4, 5–6, 7–8, 9–10 és 11–12. évfolyamokra. Ebben a tantervben megtaláljuk az egyes műveltségterületekre fordítandó időarányokat is (Pl. Informatikára 7–10. évfolyamon 6–10%, azaz heti 1,5–3 óra). Ugyanakkor a műveltségterület céljai között megjelenik az IKT eszközök más tanórákon történő használatának elvárása, általános elvárás a Digitális kompetencia fejlesztése.

Informatika műveltségterülete tartalmában jelentősen bővül:

„A fejlesztési feladatok szerkezete

1. *Az informatikai eszközök használata*
2. *Informatika-alkalmazói ismeretek*
   - 2.1 *A gyakorlati életben használt legfontosabb írásos formátumok gépi megvalósítása, igény a mondanivaló lényegét tükröző esztétikus külalak kialakítására*
   - 2.2 *Adatbázisok, adattáblák alkalmazása, keresés az adatbázisban*

3. Infotechnológia (problémamegoldás informatikai eszközökkel és módszerekkel)

3.1 Az adott probléma megoldásához szükséges módszerek és eszközök kiválasztása

3.2 Algoritmizálás, adatmodellezés (a hétköznapi életben és az iskolában előforduló tevékenységek algoritmizálható részleteinek felismerése és különféle formákban történő megfogalmazása)

3.3 Egyszerűbb folyamatok modellezése, a paraméterek módosítása

4. Infokommunikáció

4.1 Információkeresés, információközlés

4.2 Információs technológián alapuló kommunikációs formák

5. Médiainformatika

6. Az információs társadalom

7. Könyvtári informatika

8. Az elektronikus vásárlás szerepe a XXI. században”

A bejövő új témák az Internet terjedésével, a Sulinet program indításával kapcsolatosak, emellett magához vonja az előző NAT más műveltségterülethez kapcsolódó területeit.

Az Életvitel és gyakorlati ismeretek tanterve elrejti a gyakorlati tevékenységet, visszatért a témák elméleti megközelítéséhez: Munkakultúra, Háztartáskultúra, Egészségkultúra… Az Információs kultúra tartalma figyelemre méltó:

„Kommunikációs rendszerek, információs rendszerek, modellezés, mérés, vezérlés és szabályozás, automatizált rendszerek, robotika, technikai újdonságok befogadása”

Azaz az Informatika műveltségterületben minden benne van, amihez valami köze van az informatikának, kivéve az Információs kultúra – ami tartalmát illetően leginkább a mérnökinformatika alapjait jelenti – az Életvitel és gyakorlat műveltségterülethez tartozik. Az Informatika többi műveltségterülethez való viszonyát „kereszttantervi ajánlások”-kal formálták [92], ezzel – lényegében tantárgyi honlapok és szoftverek listájával – biztosítva az oktatás korszerűsítéséhez a módszertani alapokat. Az Életvitel és gyakorlati ismeretek műveltségterület oktatása azonban számos nehézséggel küzdött. Tanári oldalról a sokféle témakör megfelelő szintű ismerete lényegében lehetetlen volt, ráadásul nem jutott idő mindegyik témára. Ezért nagymértékű a témák közötti szelekció. Korábban a fiúknak és a lányoknak eltérő gyakorlati foglalkozás volt, ennek a tantervnek a kivitelezésében a tanár motiváltsága, érdeklődési területe határozta meg a tananyagot. További problémát jelentett, hogy a gyakorlathoz az eszköz (bicikli, fúró,

oszcilloszkóp, sütő) és nyersanyag (fa, fém, ellenállás, fonal, festék, liszt) nem volt biztosítva. Ezért sokszor elméleti lett a gyakorlati oktatás.

A 2012-es NAT-ban [67] az Informatika műveltségterület tovább bővült, egyben strukturáltabbá is vált.

„A fejlesztési feladatok szerkezete:

1. *Az informatikai eszközök használata*
2. *Alkalmazói ismeretek*
   - 2.1 *Írott és audiovizuális dokumentumok elektronikus létrehozása*
   - 2.2 *Adatkezelés, adatfeldolgozás, információmegjelenítés*
3. *Problémamegoldás informatikai eszközökkel és módszerekkel*
   - 3.1 *A probléma megoldásához szükséges módszerek és eszközök kiválasztása*
   - 3.2 *Algoritmizálás és adatmodellezés*
   - 3.3 *Egyszerűbb folyamatok modellezése*
4. *Infokommunikáció*
   - 4.1 *Információkeresés, információközlési rendszerek*
   - 4.2 *Az információs technológián alapuló kommunikációs formák*
   - 4.3 *Médiainformatika*
5. *Az információs társadalom*
   - 5.1 *Az információkezelés jogi és etikai vonatkozásai*
   - 5.2 *Az e-szolgáltatások szerepe és használata*
6. *Könyvtári informatika”*

Ezen belül a robotika is „átkerül” az informatikához: *„robotvezérlési grafikai feladatok megoldása fejlesztőrendszerrel”*, ami kevesebb, mint ami Életvitel és gyakorlati ismeretekben volt.

A tananyag bővülése mellett azonban csökken a műveltségterületre fordítható időarány, 6–10%-ról 4–8%-ra, megjelenik számos nevelési feladat: erkölcsi nevelés, nemzeti öntudat, hazafias nevelés… testnevelés. A NAT előírja a tartalmi leírást tartalmazó Kerettantervek létrehozását és alkalmazását, valamint felére csökkenti (általában 10%-ra) a helyi tantervi szabadságot. A NAT-ban megfogalmazott megkötéseknek számos káros hatása van. Például a 7.§ 3. pontja:

„(3) Ha az iskola emelt szintű oktatást szervez, az emelt szintű oktatásban érintett évfolyamokon és tanulócsoportokban a) idegen nyelv, matematika, magyar nyelv és

irodalom, továbbá nemzetiségi nyelv és irodalom esetén legalább heti öt, b) minden egyéb tantárgy esetében legalább heti négy tanórai foglalkozást kell biztosítania."

A minimum heti 4 óra azt jelenti, hogy a heti időkeret 11–14%-ában kell egy tantárgyat tanítani. A tanterv alkalmazásában a maximalizált 10%-os helyi elosztás lehetetlenné teszi egy évfolyamon olyan tantárgy emelt szintű oktatását, amelyikből normál szinten nincs hetente legalább 2 óra.

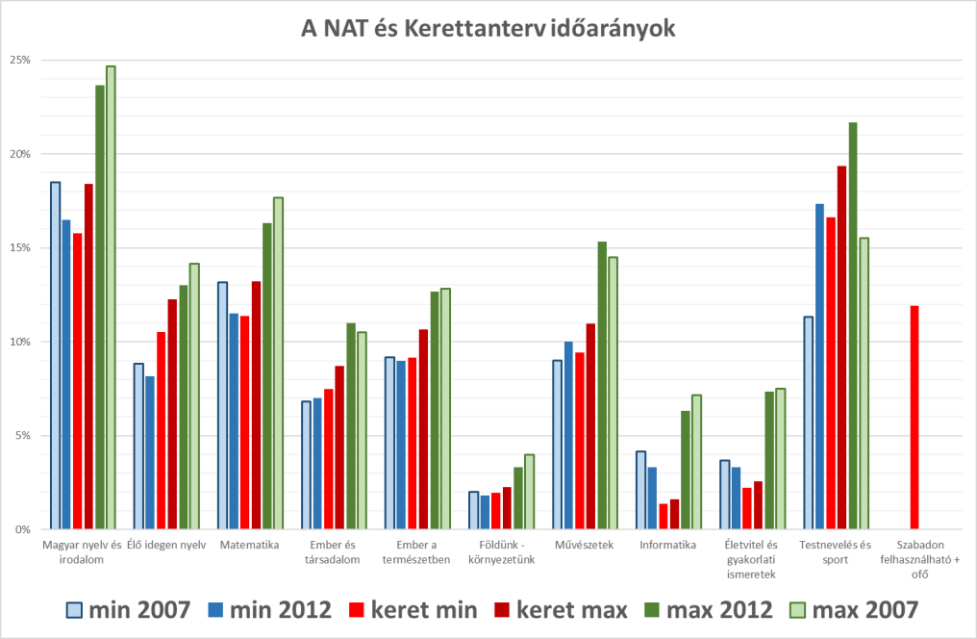
Az egyes műveltségterületek arányainak meghatározásánál a minimumok összege 90%. Ha a műveltségterületekre megadott minimumot lefelé kerekítve határozzuk meg az óraszámot, akkor az előírtnál kevesebb időt kap a műveltségterület, ami konfliktust okoz a tanítás megszervezésében. Ha mindegyik műveltségterületnél az alapóraszámból és a minimális időarányból számított értéket felfelé kerekítjük, akkor az egy hétre számított összes óraszám meghaladja a törvényben megengedettet. (4. táblázat)

4. táblázat: Minimális időarányokból számított óraszámok összesítése

Évfolyamok	**5–6.**	**7–8.**	**9–10.**	**11–12.**
Magyar nyelv és irodalom	5	4	4	4
Idegen nyelvek	3	4	5	5
Matematika	4	4	4	4
Ember és társadalom	2	4	3	4
Ember és természet	2	5	6	4
Földünk – környezetünk	1	2	2	0
Művészetek	3	3	3	3
Informatika	2	2	2	2
Életvitel és gyakorlat	2	2	2	0
Testnevelés és sport	6	5	5	6
Maximális megengedett óraszám	**28**	**31**	**35**	**35**
Számított óraszámok összege	**30**	**35**	**36**	**32**

A 11–12. évfolyamon van választási lehetőség, mert ott az arányszámok minimum értékeinek összege – tekintettel a fakultációkra – csak 78%. Azonban két fakultáció választása 4 tanórát jelent, amivel a diák óraszáma már túllépi a megengedett 35 órát, nem beszélve arról, hogy egyes egyetemi szakokra – tapasztalat alapján – három tantárgyból célszerű fakultációt választani. Például mérnökinformatikus képzés esetén matematika, fizika, informatika, orvosi képzésekre biológia, kémia, matematika (+ esetleg latin).

A 2012-es NAT [67] és Kerettanterv [69] a korábbihoz képest minden műveltségterülethez több tananyagot rendelt. Pontosabban, azzal, hogy a kompetenciák helyett a tananyag került központba, kötelezővé tett korábban választható tananyagokat. A túlméretezés kivédésére két műveltségterület idejének egy részét más tantárgyakba integrálták.[45] Így az Informatika és az Életvitel és gyakorlati ismeretek műveltségterület az arányoknak megfelelő óraszámnál kevesebb órát kaptak. Az alábbi diagramon (28. ábra) az időarányokat és óraszámokat 12 évfolyamra, műveltség-területenként átlagolva láthatjuk.



28. ábra: A 2007-es és 2012-es NAT, valamint a 2013-as Kerettantervben a műveltségterületekre meghatározott időarányok

A világoskékkel jelölt 2007-es NAT [66] minimum és világoszöld maximum között található a sötétebb árnyalattal jelölt 2012-es NAT-ban [67] megadott minimális és maximális időarány, ezen belül piros színnel jelölve láthatjuk a kerettantervi [69] óraszámoknak az évfolyamra megengedett maximális óraszámmal számolva az arányokat; minimumként a szabadon felhasználható órakeretet külön feltüntetve, maximumként a szabadon felhasználható keretet arányosan szétosztva a műveltségterületek között.

Az élő idegennyelvnél látható, hogy mindkét idegennyelv több, mint minimális órát kapott. Az Ember és társadalom műveltségterületet növeli az Etika és Erkölcs is; a Testnevelés és sport

[45] Az eredményt tekintve úgy tűnik, az a műveltségterület esett áldozatul, ami sok kapcsolattal rendelkezik a többi műveltségterülettel, valamint 20–40 évvel ezelőtt (amikor napjaink döntéshozói képződtek) még nem létezett, vagy teljesen más tartalommal létezett.

a „mindennapos testnevelés” bevezetésével jelentős többletet kapott a 2012-es NAT-ban és Kerettantervben. Jelentősen kevesebb órát kapott a Kerettantervben az Informatika, a minimum 1/3-át, illetve az Életvitel és gyakorlati ismeretek, a minimum kb. 2/3-át.

VII/2. Tananyagsűrítés

Az Informatika-Számítástechnika Tanárok Egyesülete 2015-ben végzett kutatást a szétosztott Informatika hatékonyságáról [76].

A kutatás legfőbb megállapításai:

- *Többségük[46] nem tudott különbséget tenni az informatika témakörök megtanítása és az informatikaórán már megszerzett tudás alkalmazása, kamatoztatása között.*
- *Voltak, akik kiemelték, hogy alkalmazói szinten ugyan kezelik az IKT eszközöket és igyekeznek lépést tartani az újdonságokkal, de nem rendelkeznek az informatika témakörök megtanításához szükséges kompetenciákkal (ismeretekkel, képességekkel, attitűdökkel). Nem is tartják saját feladatuknak ezeket.*
- *A témán mélyebben gondolkodó pedagógusok azonban megfogalmazták, hogy a tanfolyamokkal kizárólag saját IKT módszertani kultúrájukat fejleszthetnék, ami kétségtelenül elengedhetetlen, de ezzel nem oldódik meg a tantárgyukba áthelyezett informatika tartalmak informatika szakmódszertan szerinti megtanítása.*

A kutatás a teljes kudarcot mutatja, azonban kérdés a kudarc oka. A probléma gyökere abból a populáris nézetből adódik, hogy a mai gyerekek már az információs társadalomba születtek [5], ezért nem kell tanulniuk az informatikát; azt kell nekik is megtanulni, amit szüleik, nagyszülőik tanultak; bár látható, hogy az informatika egyre fontosabb, de elegendő, ha „a gyerek többet tud nálam”. Ez az állapot az Informatika műveltségterületen könnyen előfordulhat, de nem jelenti azt, hogy az ismeret többlet releváns tudás lenne és azt sem, hogy elegendő lenne [6].

A LAU-modell alapján: az egyes LAU-ok megtanulásához idő kell. Az, hogy egy LAU-t áthelyezzük egy másik tantárgyhoz, nem feltétlenül jelenti a hozzá szükséges idő csökkenését. A LAU fázisait tekintve, a LAU-okra fordítandó idők alapvetően összeadódnak, kivéve az 5c, esetleg 5b fázist, ahol komplex – más tantárgyon belül történő – felhasználással lehet időt

[46] A megkérdezett, nem informatika szakos tanárok többsége.

nyerni. Az informatikából tanult, legalább 5b szinten elsajátított ismeret használható más LAU-ban, más tantárgyban. Így az adott LAU-ban is előrelépést lehet elérni és **egyidejűleg** az informatikai ismeret is megerősítést nyer, elmélyül.

A kutatás adataiból látható, hogy az Informatika műveltségterület más tantárgyakon belüli tanítása a más műveltségterületek oktatói számára csak az egyes LAU-ok 5c szintjére vonatkozik. Ahol közös projektek keretében alkalmaznák az informatikát, például egy adott tárgyból végzett gyűjtőmunka, prezentációkészítés, szimulációs program vagy robot készítése lett minősítve mindkét tantárgyból, ott jellemzően egy-egy informatikai LAU-nak az 5b fázisú alkalmazására volt lehetőség. Ehhez azonban az egyes tantárgyakon belül a készség ⌊1-5b részét valahol meg kellett volna tanítani, amire az egyes tantárgyaknál se módszertani képesség, se idő nincs, ezért azt a maradék informatikaórákon kellene teljesíteni.

Az arányokkal számított óraszám 1/3-ába tömörített informatika tananyag egy olyan kerettantervet eredményezett, amelyben az 1-1 órára tervezett tananyag a tervezett háromszorosa. A tantervek, köztük a kerettanterv LAU szerinti elemzése a „Why Can't I Learn Programming?" The Learning and Teaching Environment of Programming cikkünkben [77] olvasható. Megállapítható, hogy a kerettantervben néhány témakör esetén a LAU 1. fázisra sincs idő, a LAU 2. fázisa a házi feladat. Az eredmény az, hogy a közoktatásból kikerülő diákok az 1995 előtti időszakra jellemző tudással rendelkeznek.

Példa a tanítási gyakorlatból: 2018. április, szövegszerkesztés 1. dolgozat 7. osztályban gépelésből és bekezdés- és karakterformázásból.

> Feladat egy meghívó levél írása. A diák munkája tele felesleges szóközzel, enterrel, de a lap háttérszíne kék, a betűk fehérek. Tudja, hogy melyik funkció hol van, de a felhasználás nem célszerű, az ismeret nem hasznosul.

Az informatikai ismeret egyik tantervben sem azt jelenti, hogy a tanuló ismeri a menüt. A diákoknak a szövegszerkesztés alapjait azért kell tanulniuk, hogy lehessen arról beszélni, hogy mi a célja a formázásnak, hogyan kell hatékonyan (rövid idő alatt, könnyen kezelhető) jól kinéző, a lényeget kiemelő levelet írni, plakátot készíteni. Heti 1 órában, még házifeladat mellett is, a tanultak nagyrészét a diákok elfelejtik, túl hosszú a LAU 4. fázisa. Néhány kattintás helyett hosszan piszmognak a megoldással. Bár egyszerűnek hangzik, ahhoz a néhány kattintáshoz át kell látni a feladatot, ismerni kell a rendelkezésre álló eszközöket és optimális megvalósítási utat kell találni. Eszköz ismeret, megoldási algoritmus, problémamegoldás – informatikai gondolkodás szükséges a szövegszerkesztés ⌊5b szintű ismeretéhez.

Az informatika műveltségterület célja az informatikai gondolkodás fejlesztése. Ez a gondolkodásmód – például a modellalkotás és absztrakciós készség – használható igazán hatékonyan a többi műveltségterület művelése során is, ami nem csak a hatékonyabb alkalmazáshasználatban, hanem tanulási módszerben is, kompetenciafejlesztésben is jelentős lehetne. Bár a NAT – lényegében kezdetektől fogva tartalmazza ezt a célt, a megvalósításhoz szükséges eszköz vagy idő nem biztosított.

VII/3. Tantárgyi integrációs tapasztalatok Informatikából

A már hivatkozott [76] kutatás beszámolója alapján látható, hogy nulla időkerettel nem lehet integrálni az informatikai tartalmakat a többi tantárgyba. Azonban vannak olyan országok, ahol nincs informatika tanóra, hanem integráltan tanítják [79] az informatikát és van, ahol választható az informatika tantárgy [88] [91]. Valójában kísérleteznek az egész világon, hogy hogyan oldható meg az informatikai tartalmak tanítása. Ezen belül is meg kell különböztetni legalább két területet: a napjainkban leginkább Digitális írástudásnak nevezett alkalmazói ismereteket (a 2012-es NAT Informatika műveltségterület tartalmai közül az 1., 4., 5., 6. és 2.1-es témakör) és a Programozást (2.2 és 3. témakör), amelyeknek integrált vagy tantárgyként történő oktatása további diszkussziót jelent. Az, hogy az integráció mennyire sikeres, számos tényezőn múlik. Néhány ezek közül:

1. Milyen az eszközellátottság.
2. Az informatika egyes területeit oktató tanárok mennyire felkészültek informatikából.
3. Milyen oktatási formák valósíthatók meg.
4. Melyek a konkrét integrálandó témakörök.

Az integrációs lehetőségekről az Oktatásfejlesztő Intézet 2009-ben készített tanulmányt [92], amely az OKI IIK[47] obszervációs kutatása 2005-ös adatait használja, igen tanulságos. A témáról és a kapcsolódó témákról a Jelentés a Magyar Közoktatásról sorozat [95] egyes cikkeiből tájékozódhatunk. A szándékokat, terveket a Nemzeti Infokommunikációs (Internet Társadalom) Stratégiában [96] fogalmazták meg („Az infokommunikációs oktatás felülvizsgálata és újrapozícionálása”; Internetelérés és eszközök biztosítása, e-kereskedelem, informatikai életpálya lehetőségeinek bemutatása).

[47] Országos Közoktatási Intézet Iskolafejlesztési és Integrációs Központja

Tegyük fel, hogy vannak megfelelő eszközök, oktatóanyagok úgy, ahogy a stratégiában leírták. Tegyük fel, hogy minden tanár egyben informatikatanár is, azaz ért nem csak az informatikához, hanem annak az oktatásához is, sőt, megfelelő idő is rendelkezésre áll. Vizsgáljuk meg, hogyan lehet egyes informatika témaköröket integrálni. Ezzel kapcsolatban már eddig is rengeteg ajánlás, kísérlet jelent meg, de jellemzően nem az informatika, hanem a befogadó tantárgy oldaláról közelítve a kérdést. Ilyen volt az 1995-ös NAT-hoz készített kereszttantervi ajánlás [93], a Microsoft szerkesztésében megjelent 101 Ötlet Innovatív Tanároknak [94] is, de a legújabb kerettantervekben is szerepelnie kell a tantárgyi kapcsolatoknak.

Mivel a 2012-es NAT-ban és 2013-as kerettantervben az informatika más tantárgyakba történő integrálása nem történt meg, vizsgáljuk a NAT 2018-ban kiadott tervezetét [97]. Ez a tervezet megadja a tantárgyak óraszámait is, azonban rengeteg belső ellentmondást tartalmaz a régi, 19. századi szemlélet és tananyag átmentésének ötvözése a modern oktatási igényekkel. A tárgyi tudás többszáz éve folyamatosan bővülő elvárásait fel kellene váltania az idők folyamán kikopott kompetencia-fejlesztésnek. A tantervből azonban az látszik, hogy nem váltásról, hanem kiegészítésről van szó, időtöbblet nélkül. Továbbra is merev tantárgyi keretek közé van szorítva az oktatás, miközben elvárt a tantárgyközi kooperáció – időkeret nélkül vagy olyan időkeretekkel, ami már a szaktantárgyi oktatásra le van foglalva (ilyenek a témahetek, a projekthetek, szakmai napok). A tervezett informatikaóraszámok jóval magasabbak, mint a 2013-es kerettantervben megadottak, de a tananyaghoz képest szűkösek. Jól látható a tantárgyi integráció szükségessége. A 2020-ban megjelent NAT [68] a tervezethez képest egyes tantárgyakból megnövelt tényanyagot tartalmaz, ami a belső ellentmondásokat tovább növelte, ezzel gátolja a modern oktatási paradigmák érvényesülését. Az informatika területén, illetve más tantárgyakkal való kapcsolatában a NAT nem változott a tervezés után. A leírt elvárások alapján elemezhetők az informatika egyes témáinak más tantárgyakba integrálásának lehetőségei. Az informatika oktatására szánt tantárgy neve Digitális kultúra, az informatikai tartalmak négy fő területbe lettek sorolva:

1. Az informatikai eszközök használata
2. Digitális írástudás (szöveges, rajzos, táblázatos dokumentumok, internetes kommunikáció)
3. Problémamegoldás informatikai eszközökkel, módszerekkel (összetett problémák megoldása, algoritmusok, adatmodellek, adatbázisok, táblázatkezelés, robotika, programozás)
4. Információs technológiák (webes- és mobiltechnológiák)

5.1

Az informatikai eszközök használata (1.)

Első közelítésben mondhatjuk, hogy az eszköz használatát tanítsa az, aki először használtatni szeretné az eszközt. Sőt… Napjainkban az eszközök kellően felhasználóbarátok ahhoz, hogy használója önállóan megtanulja használni. Ez a nézet számos esetben jól használható. Például számos játék, a mobil és a számítógép, a monitor használatbavétele informálisan is tanulható. Ráadásul ezen esetekben az előzetes tanulás értelmetlennek tűnik, mert mire szükség lenne rá, a tanuló elfelejti a tanultakat. A LAU modell alapján, ezen tartalmak külön időkeretben tanulása esetén a LAU^{L}4 túl hosszú lenne. Összefoglalva: az eszközök használatát a megfelelő LAU-okba beágyazottan kell (lehetne) tanítani. A beágyazott LAU azonban nem lehet nagy.

A tananyag tanórákra bontásának elveiből, a tanórai tevékenységek felosztásából következően, egy beágyazott LAU a tanórai tevékenységek szervezéséhez tartozik. Erre jellemzően 5 perc áll rendelkezésre. A fő LAU tanulásához szükséges eszköz használatának bemutatását így 5 percben kell megoldani. Amint ennél hosszabb idő szükséges, a „betanítás” felborítja a fő LAU menetét. Az informatikai eszköz használata kontraproduktív, ha az eszköz használata közben derülnek ki a problémák.

Az egyik jellemző probléma az adatok kezelése szokott lenni. Például, egy biológiaórán meg lehet tanítani a mobiltelefonnal a fényképezést, esetleg még a fénykép feltöltését is. Azonban kevés az idő arra, hogy a képek kezelését, a tárhely karbantartását, képformátumokkal kapcsolatos minőségi kérdéseket is a biológiaórán tanulják meg a diákok. Az informatikai eszköz használatának megtanulása az informatikához tartozó nagy egység, az adat létrehozása, tárolása, továbbítása témának egy esete. A biológiaórán tanult fényképezés lehet az adatkezelés^{L}0–1 része, azaz tapasztalatszerzés; illetve az adatkezeléssel kapcsolatos ismeretek alkalmazása, azaz adatkezelés^{L}4–5c része is. Mivel jellemzően az eszköznek az adathoz való viszonyáról van szó, tipikusan informatika tanórán érdemes az ehhez kapcsolódó ismereteket és készségeket összefüggéseiben is tanítani, ezzel párhuzamosan, majd utána lesz hatékony az eszköz a biológiaórán.

A másik jellemző probléma a használat során történő tanulással az eszközök használatának rutinjával kapcsolatos. Gyorsan, a célnak megfelelően, kis befektetéssel, esetenként motorikusan kell tudni használni az eszközöket. Erre tipikus példa a gépírás. A gépírást, akár a kézírást, érdemes a használat kezdeti szakaszán módszeresen tanítani. Ezzel elkerülhetők azok a hibás beidegződések, amelyek később szervi, egészségügyi problémákat okozhatnak (ugyanez bár-

melyik másik eszköz – egér, monitor, tablet, könyv, motor, gépjármű – használatáról is elmondható). Ezt figyelembe véve kellene megszervezni a tantárgyakba, tanrendbe beépítés módját, de tantárgyi szempontból több tantárgyba is integrálható:

- A tanulásának módszertana (mozgáskoordináció, mozdulatok kapcsolása) alapján leginkább a testnevelés (és sport) műveltségterülethez tartozna.
- Az alapján, hogy a magyarnyelv és irodalom műveltségterülethez tartozik az írás és szövegalkotás, a gépírás is tartozhat ehhez a műveltségterülethez, beépülhet az általános iskola alsós tantervbe (megfelelő méretű billentyűzeten).
- Mivel számítógépes szövegalkotás, lehet Informatika tantárgyon belül egy téma.
- Gyakorlati haszna és technikával kapcsolatos ismeret lévén, része lehet a Technika és életvitel tantárgynak.
- Mivel az alapok elsajátítása után önállóan gyakorolható, fejleszthető (és fejlődik a használat során), lehet külön modulként, projektként szervezni a tanulását.

Manapság az eszközök használata önállóan is elsajátítható és erre számos példa is felhozható. Azonban az önálló elsajátítás nem mentesít a tudatosan helyes használat megtanulása alól. Más készségterületről – remélhetőleg ismert és elfogadott – példák erre: a beszéd önálló megtanulása mellett a logopédus szerepe, a ceruza fogásának és az írásnak a tanítása első osztályban vagy a kormánykerék fogásának tanítása a gépjárművezetés oktatása során. Mindegyik esetben a megfelelő szokások, automatizmusok kialakítása az, amihez idő és szakember szükséges. Bármelyik tantárgyba integráljuk az eszközhasználat tanítását, három dolgot biztosítani kell: időt a megvalósításra, szakmai hozzáértést a tanárnak és szakmódszertani képzettséget a tanárnak.

A gépírás esetén elgondolkodtató, hogy napjainkban az ügyviteli ismereteket tanulóknál a vakírás alapjai kétéves modul, heti óraszám 2 + 1, valamint körülbelül még egyszer ennyi idő önálló gyakorlás[48]. Az informatikaóra keretében helyi specialitásként tanított gépírásra heti 2-órás rendszerben 24 órát[49], illetve 72 órát[50] terveztek gimnáziumokban. Természetesen, az egyes helyeken az elérni kívánt színvonal eltérő, de mindegyikre jellemző, hogy kezdetben minimálisan hetente 2 tanóra a szükséges ráfordítási idő. Mielőtt bevezetnék a tízujjas vak gépelés

[48] Budapesti Kereskedelmi és Iparkamara, Oktatási Nonprofit Kft. Ügyviteli kerettantervi ajánlásokat fejlesztő munkacsoport: Szakképzési tantervi ajánlás 54 346 02 Ügyviteli titkár szakképesítéshez. Nemzeti Munkaügyi Hivatal (2012) https://www.nive.hu/Downloads/Szakkepzesi_dokumentumok/Szakkepzesi_tantervi_ajanlasok/word/DL.php?f=54_346_02_Ugyviteli_titkar.doc [2021.10.30]

[49] Karinthy Frigyes Gimnázium informatika tanterve http://www.karinthy.hu/pages/docs/ht/informatika.pdf [2021.10.30]

[50] Móri Táncsics Mihály Gimnázium http://tmgmor.hu/_regi_oldal/dokumentumok/2014_gepiras_2013_uj.pdf [2021.10.30]

oktatását, a tanároknak is meg kellene tanulnia vakon gépelni (ez számukra is valószínűleg 1 éven át, heti 2 órás elfoglaltság) és némi szakmódszertani ismeretre is szükségük lenne.

Mivel a billentyűzetet mindenki tudja valahogyan használni, ezért a gépírás tanulása elsősorban az egészség megőrzése és a hatékony munkavégzés érdekében fontos. Az óraszámokat, időigényt tekintve megállapítható, hogy a technika és informatika tantárgyakban nincs lehetőség az oktatásra. Megfelelő óraszám mellett legalább féléves modulként képzelhető el az oktatás, emiatt a néhány napos projektek sem jöhetnek szóba. Általános iskolában így az írásoktatás vagy a testnevelés részeként képzelhető el ennek a modulnak az oktatása.

Más eszközök használatának oktatásáról hasonló szempontok mentén lehet eldönteni, hogy oktatható-e informatika tantárgy keretén belül vagy más tantárgy részeként. A legutóbb informatikához kapcsolt téma a robotika. Ennek egy eszköze lehet egy LEGO EV3 készlet, amelynek használata a robotok építését is jelenti. Ennek tantárgyi oktatása, alkalmazása a gépírás oktatásához hasonló problémákat vet fel, mivel az építés sok időt igényel és fizika, valamint mérnöki ismeretek szükségesek hozzá.

Információs technológiák (4.) integrált oktatása

Bár ez sorrendben a negyedik téma, a más tantárgyakba integrálhatóság szempontjából az eszközök használatánál leírtakhoz nagyon hasonló. A szoftverek használatát is lehet tanórába építve tanítani. Tartalmát tekintve ebbe a témakörbe tartoznak a tanórákon használható oktatószoftverek, mobil és webes applikációk, a számítógépes mérés-kiértékelés, internetes keresés.

Az egyik legösszetettebb, oktatást segítő program napjainkban a Geogebra [98], használatának tanítása matematikaórán megoldható. Ritkábban hangzik el, hogy ennek mik az előfeltételei. Általános elvárás (matematikatanárok véleménye alapján), hogy amikorra a szoftvert használnia kell a diáknak, addigra legyen általános szoftverhasználatból több éves gyakorlata. Ez azt jelenti, hogy a diáknak van gyakorlata a webes felületek, menük, grafikai felületen megjelenő objektumok kezelésében, be tudja állítani egy-egy objektum tulajdonságát. Sok esetben ma még – tanári prezentálás során – erre nincs szükség, mert a matematikai tartalom megértéséhez elég, ha a diák a tanári prezentációt figyeli és jegyzetet készít a füzetébe, esetleg nyomtatva megkapja a képernyőképet. Jellemző, hogy a program használata egy-egy alkalommal speciális esetekre korlátozódik, a szoftvert nem kell részletesen megismerni, alkalmanként elegendő 2-3 objektum használata.

A program használatához szükséges ismeret – informatikából általánosan előképzett diákokkal – az egyes matematikatanórákon 5 perc alatt elsajátítható. Az a feltétel, hogy a diákok általánosan előképzettek legyenek, ma természetesnek tűnik, azonban nem magától értetődő. Ha

csak a mobiltelefon használatában való jártasságot tekintjük, akkor többek között a szociális körülményektől és kortól is függ ennek megléte.

A tantárgyi integrációt segíti, hogy az információs technológiák oktatása már az óvodában elkezdődhet. A csak néhány funkciót engedélyező alkalmazások helyzethez illeszkedő integrált oktatása megoldható. Azon összetett, specifikus alkalmazások használata is oktatható integráltan, amelyekben az egyes felhasználásokhoz elegendő néhány funkció ismerete. Ebben az esetben azonban a tanárnak az éppen alkalmazottnál sokkal jobban kell ismernie a szoftvert. Önfegyelmezési probléma, ha a diák összevissza kattintgat és ezért elvész a programban, amit a tanárnak pillanatok alatt fel kell ismernie és a megfelelő állapotot elő kell tudnia állítania. A tanár részéről a többlet ismeret biztosítása, a diákok eltévelygésének orvosolása nem magától értetődő, ami felveti a pedagógiai asszisztens alkalmazását.

A Geogebra [98] program rendszeres tanórai használata esetén a matematikai ismeretek fejlődésével a szoftver nagyon sok funkciója megismerhető, ez azonban nem tekinthető a szoftver átfogó ismeretének. A teljes szoftver működési logikájának megismerését valószínűleg nem lehet integráltan oktatni. Ez tükröződik abban is, hogy a matematikatanárok számára 60 órás tanfolyamokat ajánlanak a program megtanulásához. A diákoknak nincs szükségük ilyen mélységű ismeretre ahhoz, hogy a szoftver segítse a tanulmányaikat és – mivel célszoftverről van szó – később sem lesz szükségük alaposabb tájékozottságra.

Azt láthatjuk, hogy ahogy a fényképezésnél sem az eszköz használata okoz gondot tanórán, hanem az adatok tárolása, továbbítása, kezelése, úgy az információs technológiák használatában sem a konkrét oktatószoftver használata miatt, hanem az e-Világban való tájékozódás és részvétel képességének fejlesztéséhez szükséges az informatikaóra.

Digitális írástudás – informatika alkalmazói ismeretek – (2.) integrált oktatása

Egy táblázatkezelő használata is része lehet a például a matematikaórának. Lehet adatokat beírni, sorozatok tagjait számolni, diagramokat készíteni, függvényt ábrázolni… Azonban, ha valaki véletlenül ponttal választja el az egész értéket a törtrésztől, akkor a beírt számadat (pl. 1,5) helyett dátum adat jelenik meg (pl. 1.5-nek megfelel a január 5). Egyetlen ilyen eset felborítja az óra menetét. Az informatikai probléma megoldására nincs idő, miközben a tantárgyi célt sem sikerül elérni. A táblázatkezelőket irodai szoftvereknek szokták nevezni, de a lényeg inkább az, hogy általános célú szoftver. Az általános, azaz széleskörű felhasználói igények kielégítése szükségessé teszi, hogy a felhasználónak mélyebb ismerete legyen a szoftver működésének miként jéről, hogy azt igényeinek megfelelően tudja használni.

Az alkalmazói ismeret más tantárgyba integrált oktatása azokban az esetekben sikeres, amikor az ismeret jól lehatárolt, azonnal használható funkcionális részeinek tanítása (LAU^L0-3 fázis) nem igényel 5 percnél többet, valamint a felhasználás során nem adódnak kapcsolódó ismeretekkel problémák. Az ilyen jellemzőkkel bíró szoftverek egyedi célú szoftverek. Az általános célú szoftverek megismerését külön szervezett LAU-ként lehet érdemben oktatni. Az is látható, hogy egyes szoftverek – például a Geogebra –, a felhasználó igényétől függő részfunkcionalitással lehet egyedi célú vagy – például a matematikaoktatás több területén egységesen használható – általános célú szoftver. A közoktatásban azokat az általános alkalmazás-típusokat érdemes informatikaóra keretében oktatni, amelyek használata a tipikus élethelyzetekben jellemző, nem tantárgy (vagy szakma) specifikus. Az informatika tantárgyon belüli megjelenésük oka az egyes alkalmazások komplex felhasználási lehetőségei. A szoftver alapfunkcióinak használatán túl az alkalmazástípus jellemzőit (jellemző objektumait, struktúráit, eljárásait és tulajdonságait) is ismerni kell a hatékony használathoz.

Problémamegoldás informatikai eszközökkel és módszerekkel (3.) integrációs lehetőségei

Ez a témakör tartozik leginkább az informatikatudományhoz. Ennek megfelelően egyes témáit sokáig szakmai tudásként tartották számon, amelynek ismeretére az átlagos felhasználónak nincs szüksége. Az egyedi, korlátozott funkciókkal rendelkező hardver eszközök és szoftverek használatának tanítása könnyen integrálható egy-egy tantárgyba. Az általános célú eszközök és szoftverek integrálásához már nagy körültekintés kell és jellemző, hogy csak egy-egy részfunkció erejéig tanítható más tantárgyon belül a használatuk. A problémamegoldás informatikai eszközeit és módszereit a konkrét szaktantárgyi probléma felvetése előtt meg kell ismerni. Egy adott szaktantárgyi helyzetben az adott problémához szükséges eszköz vagy módszer kiválasztására a már megszerzett ismeretek készletéből van lehetőség, idő. Matematikaórán, fizikaórán vagy más tanórákon a programozást, adatelemzést csak külön modulként lehet(ne) tanítani, ennek a modulnak a tantervben és tanmenetben óraszámokkal (időtartammal) együtt meg kell(ene) jelennie.

Tantárgyi tartalmak beépítése az informatika tantárgyba

> *Science is what we understand well enough to explain to a computer. Art is everything else we do. [99]*
> *Tudomány az, amit értünk annyira, hogy elmagyarázzuk egy számítógépnek. Minden más művészet.*
>
> *(Donald Knuth)*

Az informatika tantárgy léte azért állandó kérdés még napjainkban is, mert a tanított tartalmak jellemzően más szakterületeket is érintenek. A tantárgy léte napjainkban azon alapul, hogy mely tartalmak nem taníthatók más tantárgyon belül, melyek azok a „21. századi", „digitális", „számítógépes" ismeretek, amelyeket a többi szaktantárgy készen szeretne kapni. Ez a szemlélet rendkívül megnehezíti az informatika tantárgy helyének és szerepének meghatározását, lehetővé teszi, hogy az „informatikai tartalmak"-at a felhasználói igények kiemelésével „digitális kultúra és technológia" vagy hasonló néven nevezett tantárgy tanítsa. Knuth meghatározása azért fontos, mert rámutat arra, hogy mi a kapcsolata az informatika tantárgynak a többi tantárggyal. Bár Knuth a matematika területére koncentrált, állítása a többi tudományra, így a természettudományokra, sőt, a rajzolásra is igaz. Egy rajz vizuális értéke művészeti kérdés, de egy rajz elkészítésének módja, technikája tudomány. Számítógépnek elmagyarázni azt, hogy egy művészi hatást hogyan érjen el – ehhez kell az „informatikai tartalom" tudása. Ezt a megközelítést alkalmazva a tantárgy meghatározására, megállapítható, hogy a más tantárgyak tartalmának beágyazására itt is igaz az „5 perc"-es korlát.

A korábban már említett példák közül a gépírás nem tanítható tanóránként 5 percben. Az etika (különösen kiemelve a netikettet), az állampolgári ismeretek (jogismeret, ügyintézés, részvétel a közügyekben), egészséges életmód (ergonómia, munka és pihenés szervezése) tanítása akkor hatékony, ha más tantárgyakba beágyazottan, így az informatika tantárgy keretein belül is tanítjuk. Számos esetben (például ergonómia) a más tudományokhoz köthető ismeretek előkészítés nélkül is taníthatók informatika tantárgyon belül, de gyakori, hogy informatikai tipikus megoldásoknál figyelni kell, hogy más tantárgyakban megalapozott ismereteket akarjunk „megértetni a számítógéppel". Például, a másodfokú egyenlet megoldóképlete gyakran példa az elágazások szervezésére, de csak akkor építhető be informatika tantárgyba az egyenletmegoldás, ha a megoldóképletet korábban matematikaórán már tanulták a diákok. Hasonló a helyzet a prímszámokkal, a ferdehajítással és a trigonometrikus függvényekkel, a radioaktív bomlással, az árak bruttósításával. A humán tantárgyak körében az írott szöveg tagolása és a használt fogalmak, mint a lábjegyzet, a cím és alcím, a népesség, a népsűrűség, az idővonal, a korfa, a törvény,

a jogszabály, egyes színnevek[51] vagy a számok (π) ismerete az adott informatikai felhasználás előtt szükséges, mert nincs értelme ismeretlen probléma megoldását tanítani. Másik oldalról közelítve, ha a korábban más tantárgyban tanultakat építjük be a tanóra anyagába, az a másik tantárgy számára (is) jelentős haszonnal jár, mert gyakori a LAU^5c szintű használata az adott ismeretnek. Kiemelendő, hogy ez a hatás nem csak kódolás, programozás során történő felhasználással érhető el. A szöveg tördelése és stílusokkal értelmezése, egy ábra, infografika vagy animáció elkészítése során a „számítógépnek elmagyarázzuk", hogy mit szeretnénk láttatni.

VII/4. Tantárgy és a projekt

A magyar közoktatásban (és felsőoktatásban) meglehetősen nagy a zűrzavar a tantárgyak léte és projektoktatás körül, ezért szükségesnek tartom tisztázni a tantárgy és tananyag modulok (ezen belül az informatikatantárgy), illetve a projekt viszonyát.

Az alaptanterv szempontjából a tantárgy az a Learning Activity Unit, amelynek az elsajátítását egyben, összesítve mérjük. A tantárgyon belüli kisebb LAU-ok szorosabban épülnek egymásra, szorosabb kapcsolatban vannak, mint különböző tantárgyak LAU-jai. Egy-egy tantárgy jellemzően egy-egy tudományterülethez kötődő LAU-okat foglal egybe. A tantárgyak oktatására képeznek szaktanárokat, akiknek széleskörű tájékozottsággal kell rendelkeznie az adott tudományterület módszertana és a tudományterület oktatásának módszertana területéről. A tantárgyak tanmenetében az egyes LAU-ok alulról felfelé tervezettek, rögzített a kiindulás (bemenet), amelyre építkezve határozzuk meg a **tanítás feladatait**, fejlesztendő területeket, kimeneti követelményeket.

A projekt módszer a tantárgyi tartalmak meghatározásában nagyobb szabadságot ad, a gyakorlati tevékenységek, készségek fejlesztése lesz hangsúlyosabb [100]. A projekt technika és projektoktatás módszerei 50 éve változó népszerűséggel vannak jelen az oktatásban [102, 103], ismertek a helyes alkalmazás kritériumai és a módszerben rejlő veszélyek is [101, 104, 105, 106]. A módszert a pedagógusképzésben oktatják [107].

Tanári pályám során 44 nemzetközi online oktatási projektben részt vettem. Ennek során dán, finn, görög, indiai, indonéz, maláj, szingapúri, spanyol, kollégák által szervezett és 43 országból származó tanulói csoportok részvételével megvalósuló projektekbe nyertem betekintést. A projektek mindegyike megfelelt a fenti szakmai anyagokban leírtaknak. Megvalósítása az egyes helyszíneken tanóra alatt és délután történt, a projektek időtarama 3-6 hónap volt. A

[51] Első osztályos gyerek problémája amikor a kislány haját szőkére kell színeznie: „Tanár néni, nekem nincs szőke ceruzám."

tanítási gyakorlatomban a versenyre felkészülést és a szakköri munkát egyéni vagy kiscsoportos projektként szervezem meg, a tanórai munkát is igyekszem projektként megvalósítani.

A kutatásom során a központi szabályzóknak és elvárásoknak a tudományos eredményekhez való viszonyát és a megvalósítás minőségét vizsgálom. Nagyjából azt, amit M. Nádas Mária így írt le [104 p.:60]:

„A projektoktatás beszorítása a zárt oktatás keretei közé (fából vaskarika)”.

A megvalósítás során a projekt olyan **tanulási egység**, amelyben az elvárt kimenet – a cél elérése – a meghatározó, ehhez tervezzük meg a szükséges LAU-okat, ezek sorrendjét. Azaz a projekt fentről lefelé tervezett. Fontos, hogy a projektben kitűzött cél reális legyen, belátható legyen az eléréséhez szükséges út.

Egy tantárgyon belül a megtervezett haladás teljesíthető úgy, hogy az egyes célok eléréséhez projekteket rendelünk. Azaz a tananyag elsajátítását projektként implementáljuk. A tanmenet elsősorban a tanítás terve, ezért jellemző, hogy az egyes LAU-okból csak az 1-3 fázis szerepel benne nevesítve. A tanterveknek jellemző problémája, hogy a tudás minőségi fejlesztésére sokszor nincs idő tervezve, illetve gyakorlásként, fiktív idő szükséglettel, házi feladatban jelenik meg (bővebben [78]). A projekt sok esetben annyiban ad többet, hogy a LAU^L3-5c fejlesztést tervezi, mentorálást is rendelünk hozzá. Elvi lehetőség, de a gyakorlatban csak elvétve találkoztam a fordított osztályterem (flipped classroom) módszerrel, amelyben a LAU^L1-3 otthoni, önálló munka, míg a LAU^L3–5 tanórai, tantárgyi munka. A tanulás tervezése szempontjából ez a módszer is projekt jellegű, mely 1-1 tanórát tekint a projektnek.

Az „igazi” projektek több tantárgyat foglalnak magukba, egyszerre több kompetenciaterület fejlesztését teszik lehetővé. A tantárgyköziség miatt a tanórai megvalósítás nehézkes, megvalósítása jellemzően a hagyományos tanórán kívüli tevékenységként lehetséges. Az ilyen komplex projektek akkor sikeresek, ha a felhasználandó ismeretek, kompetenciák jelentős része megalapozott, azaz legalább LAU^L2 szintű, illetve – főleg adatok esetén – ismert a keresés helye, módja. A projekt célja a tanultak kiegészítése, az új ismeretek megszerzése a régiek kombinálásával, kreatív alkalmazásával történik. Előfordulhat, hogy a projekt elvégzéséhez jelentős mennyiségű új ismeret is szükséges, akár egész LAU-ok elsajátítására is szükség lehet. Ebben az esetben ennek megtanítását célszerű tanórai keretbe szervezni, egyes projektek elején felkészítő oktatást tartani. Erre láthatunk példát a RobonAut versenyre[52] készülő csapatok felkészí-

[52] BME VIK RobonAut honlapja: http://robonaut.aut.bme.hu/ [2021.10.30]

téskor, néhány közoktatási tananyagon túlmutató verseny, vetélkedő szervezése kapcsán, például a BeeSmarter[53] esetén, illetve az MTMI (STEM) témájú (robotika) versenyek felkészítő szakköreinél. A közoktatás kötelező tananyaga esetén a projekthez szükséges előképzést a tantárgyi oktatásban célszerű megszervezni, pontosabban, másik oldalról nézve: a projektet úgy kell megszervezni, hogy a hozzá szükséges alapokat mindenkinek lehetősége legyen megtanulni (29. ábra). Nem lehet hatékonyan alkalmazni – például – olyan projekteket, amelynek során a diákok egy része megtanulja a szövegszerkesztést, másik része nem, mert végül minden diákot értékelni kell a témából. A komplex projektek nagyon hatékonyak lehetnek minden területen a LAU^{L}5c fázisú ismeret elérésében. Itt ki kell emelni, hogy a hatékonyság nem a projekt teljesítésén mérhető, hanem az ismeret változásában. Azok a projektek, amelyekben már készség szintű ismereteket kell alkalmazni, még ha érdekesek is, nem hatékonyak az oktatás szempontjából. Azok a projektek sem hatékonyak, amelyek nem kapcsolódnak a tanultakhoz, mert a projekt során szerzett ismeret elkopik (30. ábra).

29. ábra: Tantervhez kapcsolódó projekt

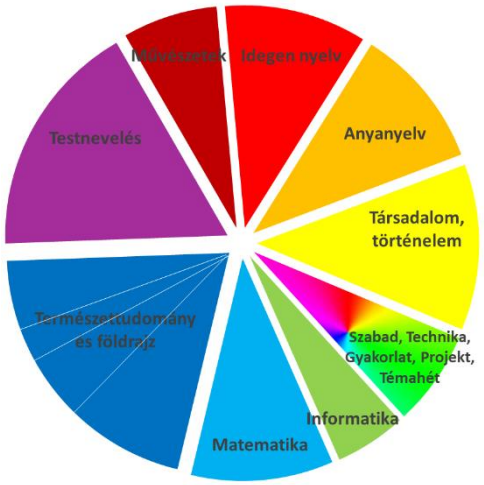
30. ábra: Önmagáért tervezett projekt

Az ábrákon az egyes színek azokat az ismereteket és készségeket jelképezik, amelyekre a projekt épül.

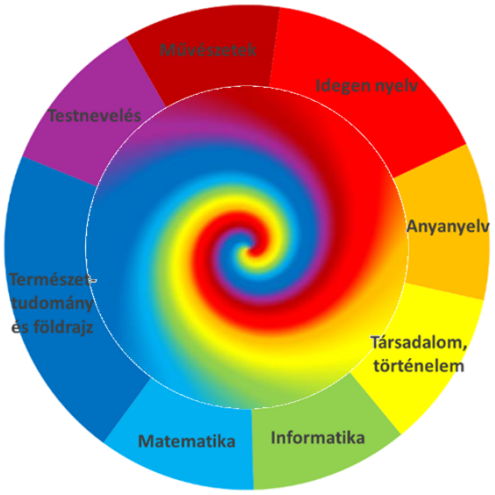
Tantárgyközisége miatt a projekteket nehéz órarendbe (45 perces tanórák) illeszteni, ezért modulként több tárgyat összevonva vagy tanórák utáni foglalkozásként lehet megteremteni hozzá az időt. Ez okozza azt a látszatot, hogy megszűnnek a tantárgyak. Valójában a tantárgyak időbeli szigorú beosztása szűnik meg. A napjainkban érvényes tantervek szigorú heti órarendi felosztása, a tanárok tanulócsoportokhoz történő egész tanévre szóló beosztása lényegében lehetetlenné teszi a projektoktatást (31. ábra) – ez a fenti idézetben a „fa”. A projektet vezető/felügyelő tanárnak nem kell rendelkeznie az egyes tantárgyak szakmódszertani ismereteivel, de

[53] BeeSmartes a PPKE-ITK PontVelünk programja részeként: https://pontvelunk.itk.ppke.hu/?node=contests [2021.10.30]

a háttérben biztosítani kell a szakmai vagy szaktanári mentorálást, a „távoktatást” vagy konzultációs lehetőséget. Ugyanakkor az eredmények minősítése, az értékelés, az egyes tantárgyak keretén belül történik (32. ábra) – ez az idézetben a „vaskarika”. Ez a fajta pedagógiai munka szintén nem tervezhető jól a jelenlegi tanári kötelező tevékenységek mellett. Megoldás lehetne (álom), ha többszakos tanárként minden szaktárgyamból a 2 órán felüli részt összevontan tudnám tanítani. A kialakított 3-4 órás idősáv alkalmas lenne projektoktatásra. Általános megoldásként ehhez hasonló lehetne, ha minden tantárgynak maximum 2 lenne az óraszáma, a felszabaduló időben „tanulószoba” foglalkozás lenne.



31. ábra:
Tantárgyak időszeletei a 2020-as NAT tervezetben[54]



32. ábra:
Tantárgyak időszeletei projektoktatáshoz[55]

Az informatikaoktatás projektje – Digitális Témahét

A közoktatásban szokásos projekt-hét általában egyik fentebb említett típusnak sem felel meg. A „projekthét” vagy „projektnapok” komplex tanulási terek, de időtartama és akció jellege miatt az éppen aktuális LAU-ok nem kapcsolódnak szervesen a tanulmányokhoz. Jellemző, hogy komplexitása ellenére az egyes projektek nem értékelhetők több tantárgyban, vagy ha mégis, akkor a jegy súlya csekély. Ezeknek a projekteknek az „egyéb” pedagógiai hasznát szokták kiemelni, lényegében a projektmódszer tanulása adja a pedagógiai értékét.

Az utóbbi 4 évben államilag is bevezetett témahetek a projekthét speciális változatai, ahol a projekt témája egy, a tantárgyi keretekből kimaradt tananyag. Két témahét esetében tematikus ismeretszerzésről van szó, a Digitális Témahét ezekkel szemben tartalmilag szabad, főleg oktatásmódszertani szempontból orientált. Megvalósítását tekintve jellemző, hogy azokat a tevékenységeket, amelyeket rendszeresen kellene végezni, a témahét alkalmával egyedi esetként,

[54] Saját készítésű ábra. A 2020-as NAT óraszámai alapján készült.
[55] Saját készítésű ábra. A tantárgyi órák és a projektoktatás integrálása.

de nagy felhajtással és publicitással végzik a tanárok. Mivel a digitális eszköz használata jellemzően nem illeszkedik szervesen az informatika tanmenethez (hiszen onnan kimaradt témákról szól) és nem rendszeres használatról van szó, a diákok számára a témahét – legalábbis az informatikai műveltség és digitális írástudásuk szempontjából – nem ad sem jelentős többlettudást sem hosszú távon használható kompetenciát.

Elrettentő példaként, 2019-ben több beszámoló bejegyzésben volt látható sikeres flashmob. A diákok (több osztálynyi) felálltak DTH (digitális témahét) formációban (vagy DTH feliratú táblával a kezükben mentek az utcán csoportban), ami legjobb esetben a testnevelés tantárgyhoz kapcsolódik és legfeljebb annyi a digitális benne, hogy egy diák egy drónt irányított és így légifelvétel készült az eseményről. „Digitális" szempontból hasonlóan problémásak azok az akciók, amelynek során a gyerekek közösen festenek egy nagy képet. A Digitális Témahét programban azért lesz ez „jó gyakorlat", mert az együttműködésről fénykép, facebook publikáció is készül. Ezzel azonban éppen a digitális kultúrára nevelés sérül, mivel ellentmondás lehet a „személyes adatok védelme" és a „mindenkinek egyenlő lehetőséget kell biztosítani" elvek között.

A Digitális Témahét a 2012-es NAT-ban lecsökkentett informatikaóraszám kompenzálásának egyik módja, amelyben az informatikai ismereteket a többi tantárgyba integrálva, a tantárgyi kereteket átlépve lehet tanítani. Az elvárások – hogy legyen dokumentált jó gyakorlat, használjon IKT eszközöket – IKT és digitális technológiai fejlődésre késztetheti a tanárokat, számukra egyfajta továbbképzést, önképzést indukálhat, de a minőség erősen kétséges.

A digitalizáció, az információs robbanás egyik következménye az informatikatudomány kiválása a többi tudományból. Ezzel együtt a közoktatásban tartalomként, tantárgyként is megjelent az informatika oktatása. De nem csak tartalmi változást jelentett, a digitalizáció és az információs robbanás szükségessé tették az oktatási módszerek és tartalmak teljes átalakítását, a kompetenciákra, kreativitásra, együttműködésre helyezve a hangsúlyt. Az ismeretközlő tanítást fel kell váltsa az ismeretszerző tanulás támogatása. Ezen módszerek egyike a projektmódszer. A projektoktatást nagyban segíti, támogatja a digitalizáció és mindkettőnek szükségszerűen be kell épülnie a közoktatásba. De a projektmódszer alkalmazásához nem szükséges digitális írástudás és a digitális írástudás fejlesztése önmagában nem követeli meg a projektmódszer alkalmazását. Az, hogy a tanév rendjében Digitális Témahét néven a tanárokat projektszervezésre „ösztönzik", hosszú távon árt a minőségi projektoktatásnak:

- Az adott hétre a programok megszervezése olyan plusz terhet jelent, ami nem terjeszthető ki a tanév többi részére, az oktatásnak ilyen módon nem válik szerves részévé.

- Az adott hétre koncentrált projektek nem illeszkednek az oktatási folyamatba, ezért a tanultak jelentős része később elfelejtődik. Nem a szükséges tartalom megtanulásáról szól a projekt, hanem a projekthez rendelnek valamilyen tartalmat.

A témahét a digitális írástudás fejlesztésében is csak minimális szerepet tölt be, kérdés, hogy jobb-e a semminél:

- Sok esetben a megvalósításban csak kényszerítve jelenik meg a digitális jelző és ezzel együtt nem biztos, hogy megfelelő formában, felhasználással. (Például: közös rajzolás, flashmob)
- Amikor digitális kompetencia fejlesztése a cél, akkor sem hatékony, mert csak minta (akció), így várhatóan csak a LAU^{L}2 szintig lehet eljutni.
- Akció jellege miatt nem lehet digitálisan teljes, mert a koncentrált eszközigény túlterhelést okoz. Hatalmas pazarlás egy rendszer kapacitását egy egyhetes csúcsidőre tervezni, ebből következően a csúcsidőben fennakadások lesznek. (Nem bírja a wifi hálózat az egyszerre csatlakozók nagy számát; 1 eszközt 3–20 diák használna…)

Összességében a fentiek alapján az a véleményem, hogy a Digitális Témahét kezdeti lépésnek hasznos, de a négy éve futó, államilag időben és tantervben rögzítve elveszítette innovációs erejét, sőt, visszatartja a projektmódszer és digitális írástudás tényleges integrációját azzal, hogy kiemeli a mindennapok tevékenységei közül. Érdemes lesz megvizsgálni, hogy a koronavírus miatt átalakult oktatási gyakorlatban mennyire lesz értelmezhető a digitális eszközök használatában és a projektmunkában a koncentrált „témahét”.

5.2

VIII. Az informatika szerepe a közoktatásfejlesztésben

Oktatási stratégia, oktatásmódszertan szempontjából fontos a projektoktatás elterjedése, a kompetenciaalapú, tanulásközpontú módszerek kidolgozása. Ezzel párhuzamosan a természeti és a társadalmi környezet megismerése, az egyén és környezete viszonyának modellezése és megértése miatt ma már nélkülözhetetlen az informatika alapjainak, az informatikai gondolkodásnak és ezek alkalmazásának az elsajátítása. Az informatika nem csak a tanított tartalmakon keresztül kapcsolódik a többi tantárgyhoz, hanem az oktatás módszerei, tanári kompetenciák révén is. A projektoktatás, a távoktatás, a fordított osztályterem minden formában jelentősen támaszkodik az infokommunikációs technológiákra, a digitális eszközök alkalmazására.

A 21. század informatikai stratégiái mind kiemelten foglalkoznak az oktatással. Jellemzően a főbb pontok: kell eszköz, kell tartalom és kell tanárképzés. Ebből az informatikaoktatás szempontjából a tanárképzés specifikációja sokatmondó. Az 1999-es Magyar Válasz az Információs társadalom kihívásaira [108 p: 49] alapján:

> *„Az információs technológiai eszközök használatával – szaktárgyuktól függetlenül – a pedagógiai, didaktikai, oktatástechnológiai stúdiumokhoz kapcsolódóan kell a leendő tanároknak megismerkedniük. Az oktatástechnológia egyre nagyobb mértékben át fog alakulni multimédia-ismeretté, multimédia-pedagógiává. A felsőoktatási intézményekből kikerülő fiatal tanároknak ismerni, érteni, és célirányosan használni kell az új, integrált médiát.*
>
> *A digitális taneszközök használata új pedagógiai módszereket feltételez. A tanárok kiképzése eddig technikai és nem tartalmi jellegű volt: egyre többen tudnak szöveget szerkeszteni táblázatot kezelni, de az eszközök pedagógiai felhasználásáról a továbbképzéseken nem hallanak. A tanárjelöltek képzésébe még nem épült be a számítógéppel segített tanítás és tanulás módszereinek megismerése…”*

A 2001. májusban megjelent Nemzeti Információs Társadalom Stratégia [111] ennek megfelelően a tanárok digitális pedagógiai és IKT képzését tűzte ki célul. 2016 októberében, 17 évvel később, a Digitális Jólét Program részeként jelent meg a MDOS – Magyarország Digitális Oktatási Stratégiája [112], amelynek most a végrehajtási szakaszában vagyunk. Ebben még mindig fő feladat a pedagógusok számára digitális pedagógiai és IKT kompetencia fejlesztése. A világban történt minőségi ugrásra csak az IKER (Infokommunikációs Egységes Referenciakeret) kapcsán találunk utalást, valamint ehhez kapcsolódóan a 48. oldalon a 17-es lábjegyzetben:

„digitális kompetencia, digitális írástudás, digitális műveltség nincs megfelelő mértékben definiálva Magyarországon, ezek a fogalmak nincsenek a hazai viszonyokra, fejlesztési szükségletekre szabva, meghatározva. Cél tehát, hogy állami szabályozás révén meghatározásra kerüljön a digitális kompetenciák egységes, nemzeti referenciakerete, a nyelvi kompetenciák és vizsgarendszerhez hasonló keretrendszer kialakítása révén. A referenciakeret tartalmazza a digitális kompetenciák szintjeit, az ezeken belül elvárt kompetenciaterületeket, elvárt készségeket és azok szintjei…”

Az MDOS tanterve szerint diákok, gyerekek számára kezdetektől szerepel a tananyagban a problémamegoldás számítógéppel, az adatok elemzése, a programozás. Az MDOS bevezetésében kiemelik, hogy „minden tanulónak lehetősége legyen arra, hogy elsajátíthassa a jövő technológiáit, nemcsak az informatikát, hanem a robotikát, a kiterjesztett valóságot, vagy a 3D nyomtatást”[56]. Azaz tanul informatikát, amin belül – mint azt fentebb láttuk – a problémamegoldás informatikai eszközökkel és módszerekkel témakör az, ami leginkább az informatika tantárgyat jellemzi, nem integrálható más tantárgyba. Az IKER kompetenciák között a legmagasabb, 4. szinten jelenik meg a problémamegoldás, egyszerű futtatható program készítése. [114 p: 5]. Bár a keretrendszer átdolgozása folyamatos (európai szinten is), mindenhol a digitális kompetenciák magas szintjén jelenik meg a programozás és ezzel együtt az informatikatudomány alapjainak ismerete.

Az informatika tantárgy lehetőséget ad arra, hogy a digitális kompetenciákat általános törvényszerűségeken keresztül sajátítsák el a diákok, amely ismeret alkalmassá teszi őket a későbbiekben is a digitális világban önálló fejlődésre. Ezzel szemben a tanárok csak az IKT számukra szükséges aktuális elemeire vannak felkészülve. Emiatt a digitálisan felkészült tanárok sem tudják kihasználni a diákok informatika ismereteit, mivel nem tudják, hogy mire építhetnek, illetve amit tanítanak az hogyan kapcsolódik a digitális kompetenciákhoz.

Néhány tapasztalt eset:

- Matematikából a Geogebra program használata a gyakori, de elfedi, hogy mi történik a háttérben. (Hasonló jelenség: a szorzás tanulása előtt számológép használat.)
- A programozás „változó” és „paraméter” fogalmának értelmezését egy másik absztrakciós szinten, a táblázatkezelésben tanult abszolút és relatív hivatkozás megkönnyíti, gyakorlatban alkalmazza, de ez a paraméteres egyenletmegoldásban nem jelenik meg.

[56] https://digitalisjoletprogram.hu/hu/tartalom/dos-magyarorszag-digitalis-oktatasi-strategiaja [2021.10.30]

- A diagramok értelmezése több tantárgyban szerepel, ugyanakkor a kategória-tengely és x-tengely közötti különbség nem ismert, ami hibás megoldásokat eredményez.
- Fizikából az egyenes vonalú egyenletes mozgás leírását egész más alapokon lehet kezdeni, ha előtte már találkozott a diák egy számítógépes objektum vagy robot mozgatásával: minden időegységben pozíció = pozíció + lépés.
- Rajzból kérdésessé vált, hogy az additív vagy a szubsztraktív színkeverést tapasztalta-e meg jobban a diák, ettől függően: hogyan keveri, honnan származtatja a keverék színt.
- Nyelvtanból megengedett (helyesírási szabályzat alapján helyes) az idő elválasztójelére a pont, de számítógépen kettősponttal jelöljük, a pont – a dátum elválasztója – időpont jelölésre félreérthető.

Minden tantárgyban számolni lehet azzal, hogy a változó és a paraméter absztrakciója korábbra tevődik.

Azokban az esetekben, amikor a nem informatikaszakos tanár informatikát tanít, a hatékonyságot csökkenti, ha nem tudja, hova vezet az a kis részlet, amit ő ad át. Ma már óvodában, általános iskola alsó tagozatában használják a padlórobotokat. Az óvók, tanítók tudják, hogy ezek a feladatok segítik a gyerekeket az önfegyelem gyakorlásában, a szerializálás elsajátításában, sorminták készítésében… az iskolaérettséget jelző kritérium teljesítéséhez kiváló fejlesztőeszköz a padlórobot. De, vajon milyen elképzelése van a tanítónak arról, hogy ennek a gyakorlásnak, készségfejlesztésnek milyen kapcsolata van a későbbi informatika tanulmányokkal? A tanító saját tapasztalata és szakmai képzése rálátást ad más tantárgyakkal, ismeretekkel való kapcsolatokra. Jellemző, hogy mennyire tudatosan tanítja például a ceruzafogást, a vonal és a kör rajzolását, a mennyiségeket, a tárgyak megnevezését. Mi a helyzet az informatika alapfogalmaival? Egy padlórobottal játszva megtapasztalható az adat és a ciklus fogalma, de céltudatos tanítás nélkül ez ki is maradhat. Hasonlóan, bármelyik informatikai eszköz vagy alkalmazás felhasználása egy adott területen lehet fejlesztő, de a tanár ismeretei az adott eszközről vagy alkalmazásról erős hatással van arra, hogy a diák milyen hatékonysággal (céltudatosan, kreatívan) használja azt.

A stratégiákat a digitális átállás szempontjából Racskó Réka vizsgálta doktori értekezésében. A nagyon részletes áttekintésből a pedagógusokra vonatkozó részt emelem ki [115 p: 191]:

> *„Összességében azt mondhatjuk, hogy a pedagógusok kulcsszereplők a kulcskompetenciák fejlesztésében és továbbfejlesztésében. Az is jól látszik, hogy nem az in-*

formatikatanároknak szánnak kiemelt szerepet; hanem minden pedagógusnak a saját szakterületén, illetve oktatott tantárgyában kell az IKT-integrációt megvalósítania, fejlesztve az IKT-kompetenciát és a digitális írástudást."

A tanárok saját szakterületükön és tanítási gyakorlatukban lassan 20 éve tanulják a technológiát és tanítják is, fejlesztik a tanulók technológiai kompetenciáit. IKT alkalmazásának integrálása nagy erőkkel folyik 20 éve, de az IKER szinteket tekintve, az IKT-integráció inkább mennyiségi, mint minőségi a változás.

Az MDOS-ban megjelenik az informatika ismeretének szükségessége a tanulók számára. Az elvárás alapja a nemzetközi gyakorlatok és tapasztalatok elemzése volt. A jövő állampolgárának képesnek kell lennie összetett rendszerek működésének megértésére, elemzésére, ebben saját szerepének (résztvevő, vezérlő) felismerésére; elemző értelmező gondolkodásra. Ezzel szemben sokszor az a jellemző, hogy a pedagógus értelmezés és elemzés nélkül tanulja (szedi fel) az eszközkezeléssel és alkalmazáshasználattal kapcsolatos tudnivalókat. Sok esetben a pedagógus követi a diákokat, azaz amit a diák magától megtanul, azt tanulja ő is. Az IKT-integráció tipikusan az informálisan tanult felhasználói ismeretekre vonatkozik, nem tartalmazza az ismeretek közötti magasabb szintű összefüggések felismerését.

Az elmúlt húsz év pedagógus-továbbképzési erőfeszítései és elért eredmények azt mutatják, hogy az IKT-kompetenciák mennyiségi fejlesztése hosszú távon nem eredményes. Az MDOS-ban benne van a pedagógusok IKER 4 szintű képzettségének a gondolata, ugyanakkor a használt kifejezések definíciójában azért látható bizonytalanág, mert a valódi (nemzetközi) értelmezés alapján minden pedagógusnak IKER 4. szintű informatikai kompetenciákkal kellene rendelkeznie. Minden pedagógusnak meg kellene tanulnia a közoktatási informatika tananyagot. Ez alapkészség ahhoz, hogy képes legyen követni a fejlődést.

Kollégáim körében úgy látom, hogy ez a gondolat már a jelenlegi informatikatananyaggal is elrettentő a pedagógusok számára, pedig az új, (mellesleg, finoman, éppen csak) programozást is tartalmazó 2020-as NAT-ra vonatkozóan kell értelmezni. Ezért látható az MDOS-ban a „forró kása kerülgetése", a robotika informatikatananyagként feltüntetése. Úgy tűnik, hogy az, hogy az új kompetenciák (informatikai gondolkodás, programozás) szükségesek lennének minden pedagógus számára, de ez az elvárás pedagógusok jelentős részében félelmet kelt.

5.3

IX. Az informatikaoktatás tartalma

Egy-egy téma oktatásáról cikket írtam [57, 58, 121, 130, 133, 137, 159], más témáknál beszélgetések, tanári konzultációk során mondtam el, hogy mi a különbség a hagyományos informatikatananyag és aközött, amit tanítok. A cikkek ellenére sem mondhatom teljesnek a leírást. Amire most kísérletet teszek, az a szemléletmódnak, az informatika tanításáról való gondolkodásnak a bemutatása – remélhetőleg – jellemző példákon keresztül. Ehhez a NAT 2020 Digitális kultúra tantárgyát veszem alapul, amely a legutolsó megjelent szakmai anyag. A tanterv kidolgozói nemzetközi tapasztalatokat figyelembe véve dolgozták ki az egyes témaköröket. A megjelent dokumentumot kiegészíti számos előadásuk, amely a munka hátterét mutatja be. Ezek egyike az ÉGIG[57] kezdeményezésére szervezett informatika és technika munkaközösségi találkozó, amelynek jegyzőkönyvében [127] leírva is megjelent az egyes témakörök súlyozása:

> *A nemzetközileg elfogadott gyakorlat alapján az informatikaoktatás területei: 1. Az informatikai eszközök használata; 2. Digitális írástudás (szöveges, rajzos, táblázatos dokumentumok, internetes kommunikáció); 3. Problémamegoldás (összetett problémák megoldása, algoritmusok, adatmodellek, adatbázisok, táblázatkezelés programozás); 4. Információs technológiák (robotika, webes és mobiltechnológiák). Az 1. témakör nem önállóan, hanem a másik három témakörben jelenik meg. A három fő témakör aránya a jelenlegi magyar gyakorlatban: 80%–10%–10%, míg a kívánatos az 50%–30%–20% vagy 40%–40%–20% lenne (iskolatípustól függően).*

Az informatika oktatása során tudatában kell lenni annak, hogy az informatikatudomány fejlődik, változik, újraértelmezi a fogalmakat. Az oktatás jellemző célja az adatok értelmezésének és a modellezésnek a tanítása, az informatikai gondolkodás fejlesztése. Ezért jellemzően nem definíciók, hanem kérdések, problémafelvetések sorával jellemezhető az informatikaoktatás. A feltett kérdések, problémák, értelmezési feladatok nem feltétlenül köthetők korosztályhoz, a lehetséges válaszok azonban jellemzők lehetnek egy-egy korosztályra, a válaszoló tudásprofiljára, tapasztalatára.

[57] Élenjáró Gimnáziumok Igazgatóinak Grémiuma

IX/1. Az informatikai eszközök használata (1.)

Ez a téma a NAT 2020-ban a hardverrel és operációsrendszerekkel kapcsolatos gyakorlati ismereteket fedi le. A tematikus egységhez tartozó „fejlesztési területek” és „eredménycélok”:

- *Ismerkedés az informatikai környezettel.*
  - *… megnevez néhány informatikai eszközt, felsorolja fontosabb jellemzőit.*
  - *Megfogalmazza, néhány példával alátámasztja, hogyan könnyíti meg a felhasználó munkáját az adott eszköz alkalmazása.*
  - *Egyszerű feladatokat old meg informatikai eszközökkel. … összetettebb funkciókat is alkalmaz.*
- *Gyermekeknek készített alkalmazások használata.*
  - *… választ … néhány applikációt, digitális tananyagot, oktatójátékot, képességfejlesztő digitális alkalmazást.*
  - *… használ néhány, életkorának megfelelő alkalmazást, elsősorban információgyűjtés, gyakorlás, egyéni érdeklődésének kielégítése céljából.*
  - *… feladathoz, problémához digitális eszközt, illetve alkalmazást, applikációt, felhasználói felületet választ. Felsorol néhány érvet választásával kapcsolatosan.*
- *Az informatikai eszközök önálló használata.*
  - *Célszerűen választ a feladat megoldásához használható azonos funkciójú informatikai eszközök közül.*
  - *Az informatikai eszközöket önállóan veszi használatba, a tipikus felhasználói hibákat elkerüli, és elhárítja az egyszerűbb felhasználói szintű hibákat.*
  - *Értelmezi az informatikai eszközöket működtető szoftverek hibajelzéseit, és azokról beszámol.*
- *Az operációs rendszerek önálló használata.*
  - *Önállóan használja az informatikai eszközök operációs rendszereinek felhasználói felületét.*
  - *Önállóan kezeli az operációs rendszer mappáit, fájljait és a felhőszolgáltatásokat.*
  - *Használja a digitális hálózatok alapszolgáltatásait.*
  - *Tapasztalatokkal rendelkezik a digitális jelek minőségével, kódolásával, továbbításával kapcsolatos problémák kezeléséről.*

- *Informatikai eszközök felhasználásának lehetőségei, az informatikai környezet.*
  - *Ismeri és tudja használni a célszerűen választott informatikai eszközök és a működtető szoftvereik felhasználási lehetőségeit.*
  - *Ismeri a digitális eszközök és a számítógépek fő egységeit, ezek fejlődésének főbb állomásait, tendenciáit.*
  - *Képes az informatikai környezetének tudatos alakítására. Ismeri az ergonomikus informatikai környezet jellemzőit, tevékenysége során figyelembe veszi a digitális eszközök egészségkárosító hatásait, óvja saját maga, és társai egészségét.*
  - *Képes önállóan informatikai eszközöket használatba venni és a tipikus felhasználói hibákat elkerülni, egyszerűbb felhasználói hibákat elhárítani.*
- *A mobileszközök, számítógépek, hálózatok operációs rendszerei, felhőszolgáltatások.*
  - *Céljainak megfelelően használja a mobileszközök és a számítógépek operációs rendszereit.*
  - *Igénybe tudja venni a feladataihoz szükséges operációs rendszer és a számítógépes hálózat alapszolgáltatásait.*
  - *Követi a technológiai változásokat az elektronikus könyv, hangoskönyv, oktató-videó és videóblog, az oktatást támogató portálok, alkalmazások használatával.*
- *Segédprogramok, digitális kártevők elleni védekezés, állományok tömörítése.*
  - *Önállóan használja az operációs rendszer segédprogramjait és képes a munkakörnyezet beállításainak elvégzésére.*
  - *Tisztában van a digitális kártevők elleni védekezés lehetőségeivel.*
  - *Használja az állományok tömörítését és a tömörített állományok kibontását.*

Látható, hogy a „fejlesztési területek” az eszközhasználatra vonatkoznak, az „eredménycél” pedig egy gyakorlati tevékenységre való képesség. Ezek a fejlesztési területek és célok nem rosszak, de nem biztosítják sem a jövőbeni önálló fejlődést, sem az informatikai gondolkodás fejlesztését. A megnevezett eredménycélokon túlmutató hosszútávú célok eléréséhez, az informatika tudomány szemléletmódját figyelembe véve a téma tanításakor a főbb fejlesztési területek:

- A használt eszközök, operációsrendszer és egyéb alkalmazások működésének modellezése.

- Az eszközökre jellemző adatok értelmezése.
- Az operációsrendszer és az alkalmazások adatmodelljének jellemzése, objektumok, tulajdonságaik és függvényeik tanulmányozása.
- Az *adat* és a jel (digit) különböző megjelenési, tárolási és továbbítási módjainak megismerése.
- Az eszközök és alkalmazások működési algoritmusainak tanulmányozása, modellezése.
- Informatikai gondolkodás fejlesztése, adott feladatokra és problémákra hatékony megoldások tervezése és kivitelezése, a megoldások verifikációja és validációja.
- Működési problémák beazonosítása a modellek és a jelenség alapján; hibás megoldások, hibalehetőségek felderítése.

Például:

A korábban már tárgyalt gépírás tanulás esetén az eszközhasználattal kapcsolatos fejlesztési terület annak ismerete, hogy a tízujjas vak gépírási gyakorlattal kihasználható egy ergonomikusan optimalizált környezet, begyakorolhatók a megfelelő eljárások a billentyűzet helyes használathoz. Informatikai szempontú fejlesztést jelent, ha a billentyűzet helyett a hasonló beviteli lehetőségek, például a képernyőbillentyűzet és kivetített billentyűzet ergonómiai jellemzése, az ergonómia szempontrendszere is téma.

Az ergonómia mellett, a billentyűzet – karakteres beviteli eszköz – részeként tárgyalandó fogalom az adat, a jel (digit) a karakter kódolása is.

Néhány kérdés:

Mi a szerepe a billentyűn látható karakternek, mi van, ha lekopott? Milyen kapcsolat van a billentyű jele és a képernyőn megjelenő jel között? Miért vannak nemzeti karakterkiosztások? A nyomtatott nagy ’A’ az ’Á’-tól egy jellel, a vesszővel tér el; mi a különbség a leírt karakter jelei, a billentyűleütésekkel létrehozott jelek és a tárolt karakterek között? Az egyes karakterkódolási, írási rendszerekben (pl. morze, braile írás, daktil) mi jelenti a jeleket, hogyan adható meg az ’A’, illetve az ’Á’ karakter? Mi a „bájt”, mi a „bit” és mi a kapcsolat a kettő között?

Példa:

Mi lehet a „MAR” szóból, ha 1 jelet módosítunk? Nyomtatott írásban lehet MÁR, MAP, MÄP. Binárisan kódolva, 1 bit módosításával lehet: LAR OAR, IAR, EAR, M@R, MCR, MER, MIR, MAS, MAP, MAV, MAZ.

A hibakezelés irányából közelítve a témát, néhány példa:

A digitális írástudás birtokosa képes kell legyen legalább egy elmélete arra vonatkozólag, hogy a prezentációjának alsó része miért nem jelenik meg a kivetítőn. Vagy, hogy miért nincs hangja a hangszórónak, miért nem jelenik meg a leütött billentyű karaktere a képernyőn, miért tarthat egy adatművelet hosszú ideig.

A használt hardver-szoftver rendszer működési elveinek, az adattárolás és továbbítás módszereinek ismerete alapján egy hibajelenséget (vagy nem várt jelenséget) tudni kell diagnosztizálni, majd döntést hozni a szükséges lépésekről. Ehhez ismerni kell az eszközök, az operációsrendszer és az alkalmazások működési elvét, készség szinten kell alkalmazni az informatikai gondolkodást.

A konkrét eszközök és alkalmazások az informatikai szempontú megismerésén túl, az elméleti összefüggésekre is ki kell térni, mert a fejlődés új rendszerek önálló megismerésének képességét teszi szükségessé. Ezért fontos a használt fogalmak elemzése, jelentéseinek megismerése.

Például:

Mit jelent a „digitalizáció”? A világ digitális vagy analóg? A billentyűzet hardvereszköz? Mit csinál a CPU? Mi az és hogyan működik az ALU? Hol vannak a fájlok fizikailag és mi a kapcsolat a fizikai és logikai adatstruktúra között? Ha egy gép (szerver vagy munkaállomás) lehet egyetlen fájl, akkor hogyan értelmezhető ennek a gépnek az operációs rendszere és a gépben látszó eszközök? Mi a kapcsolat a valódi gép és a rajta futó virtuális gép között?

Az Informatikai eszközök használata a legkisebb témakör, amely a többi témakörbe integrálva tanítandó. Óraszám nem tartozik hozzá, ezért az oktatás során külön figyelmet kell fordítani az informatikai kérdések felvetésére, az ismeretek elsajátítása során az informatikai gondolkodás fejlesztésére. A NAT eredménycéljaiban olvasható „használ” és „választ” mellett az informatika tudáshoz a „modellez”, „tanulmányoz”, „tervez”, „felderít” cselekvések vezetnek.

6.2.1

IX/2. Információs technológiák (4.)

A legkisebb súlyú, a tanórák 20%-ára tervezett témakör az Információs technológiák téma. Negyedik témaként ide sorolnak mindent, ami az előzőkbe nem fért bele. Főbb területek: célszoftverek használata, kommunikáció, netikett, biztonság, együttműködés eszközei, szocializáció az

információs társadalomban. Jellemző eredménycél, hogy a diák az adott témát „ismeri” és lehetőséget helyesen „használja”. Informatikai szempontból az lenne kívánatos, hogy az adott témákról és lehetőségekről rendszerszintű ismeretei legyenek, értse a használat hátterében zajló folyamatokat, a szabályok megsértéséből származó károkat és következményeket.

A témakör rendkívül szerteágazó, a megnevezett fejlesztendő területek gyűjtőfogalmak. Egyetlen fejlesztendő terület eredménycéljait tekintve is látszik a célok megfogalmazásának szempontja:

- *Információkeresés és online kommunikáció*
  - *Ismeri és alkalmazza az információkeresési stratégiákat és technikákat, a találati listát a problémának megfelelően szűri, ellenőrzi azok hitelességét.*
  - *Etikus módon használja fel az információforrásokat, tisztában van a hivatkozás szabályaival.*
  - *Használja a két- vagy többrésztvevős kommunikációs lehetőségeket és alkalmazásokat.*
  - *Ismeri és szükség esetén alkalmazza a hátránnyal élők közötti kommunikáció eszközeit és formáit.*
  - *Az online kommunikáció során alkalmazza a kialakult viselkedési kultúrát és szokásokat, a szerepelvárásokat.*

A fenti célok eléréséhez kapcsolódó informatikai kérdések:

- Az információkeresést segítő eszközöknek hogyan épül, frissül a keresési tartománya?
- A kapott találati lista mennyire teljes?
- A találati lista rangsorolásának mik az elvei?
- Milyen mértékben befolyásolja az ítélőképességet a véleménybuborék?
- Mi hitelesít egy adatot, dokumentumot, mire vonatkozik a hitelesítés?
- Milyen működési, társadalmi, illetve személyes problémák, konfliktusok elkerülését szolgálják az etikai szabályok, viselkedési kultúrák és szokások?

A többi fejlesztendő területre is jellemző, hogy az eredménycélok az éppen aktuális információs technológia alkalmazására irányulnak. A leírt cél az, hogy ismerje a technológia alkalmazásának lehetőségét, az alkalmazás szabályait. Az informatikai tudás ezen felül azt jelenti, hogy az egyén érti a szerepét az információs társadalomban, tisztában van azzal, hogy *rendszerkomponense*t használ, amivel ő is a rendszer egy komponensévé válik. Érti a használt

technológiák *működési elv*ét, helyesen *értelmezi* a kapott *adatokat*, átlátja, hogy az egyes technológiák használata során milyen adatokat szolgáltat önmagáról, fel tudja mérni, hogy ennek milyen hatása lehet másokra, illetve önmagára nézve.

6.2.2

IX/3. Digitális írástudás (2.)

A legdominánsabb témakör, amely a tervek szerint az informatikaórák 40–50%-át fedi le. Az egyes fejlesztendő területek jellemzően egyes dokumentumtípusokhoz köthetők. Az eredménycélok itt is a felhasználási lehetőségek ismeretét jelentik.

A digitális írástudás – a témakörök alapján olyan készség, amely – a gondolataink, az érzéseink kifejezésére ad lehetőséget. A „digitális írás" során digitálisan tárolt dokumentumot hozunk létre, amelynek olvasója, értelmezője – alapértelmezetten – humán egyed. A dokumentumok eszerint az ember-ember közötti kommunikáció csatornái. A digitális írás a gondolat kódolása. Az egyes dokumentumtípusok használatának ismerete olyan, mint egy adott nyelven a beszéd képessége, amely azonban a nyelvtan és beszédstílus, esetenként a metakommunikáció ismerete és használata nélkül zajos, pontatlan kommunikációt eredményez.

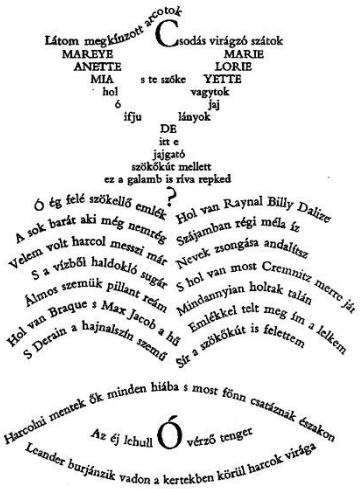
Az ember ötféle érzékelésre képes: látás, hallás, tapintás, ízlelés, szaglás, de a külvilággal több, mint 80%-ban látás útján van kapcsolata [128 p:14] (ha sérült, akkor a hallás és tapintás válik dominánssá). A digitális világgal – illetve azon keresztül más emberekkel – csak látás és hallás útján van kapcsolatunk, ezért a dokumentumokban csak vizuális és audió adatokat tárolunk, de a vizualitás elsődlegessége nagyon jellemző. Ember-ember közötti kommunikáció során az adó a gondolatait, érzéseit, állapotát kifejező adatokat „küld", amelyek – tekintettel a vevő tulajdonságaira – főleg audió-vizuálisan dekódolhatók. Említésre méltó, hogy a nyomtatást követően tapintással és szaglással is kaphatunk információt.

Az ember egyéni és társadalmi evolúciójában jelentős szerepe van a beszéd (artikulált hang), majd az írás, nyomtatás kialakulásának. A digitális írástudásnak is hangsúlyos témája a szöveg rögzítése. Azonban a beszélt nyelv és írás soha nem tudja önmagában pontosan kifejezni a gondolatot, érzést, állapotot. A pontosítást például metakommunikációval, intonációval érhetjük el. A digitális írás során a szöveg karakterek képeként és karaktersorozatként is értelmezhető, amelynek információtartalmát kiegészíti a szöveg digitalizálásának módja, a karakterek elrendezésének módja (például a szöveg tördelése, kapcsolása), a vizuális megjelenést befolyásoló tulajdonságok és a hozzáadott nem szöveges – de jellemzően vizuális – elemek.

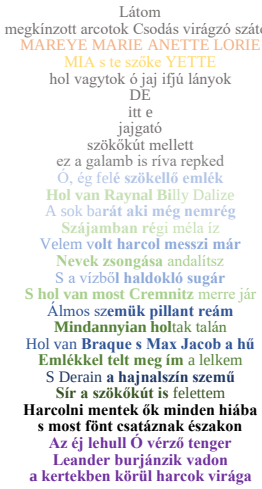
Példaként a digitális írás módjaira és ezek kommunikációs szerepére nézzük Apollinaire versét, amelynek szövegét Radnóti Miklós fordította.

- A szöveg szavalható, ami digitálisan rögzíthető. A tartalom alapján a hangdokumentumhoz háttérzaj (vagy zene) adható.
- A mű eredetileg képvers, digitalizált formája rasztergrafikus kép, a vers szövege is képi elemként jelenik meg. (33. ábra)
- A szöveget karaktersorozatként tárolva, a tördelés és tipográfia elemeit használhatjuk további információ közvetítésre. (34. ábra)
- A szöveget kifejezésekre, szavakra szedve vektorgrafikus algoritmusokat használva szófelhőt készíthetünk belőle. (35. ábra)

5. táblázat: Apollinaire – Radnóti Miklós: A megsebzett galamb és a szökőkút



33. ábra: képvers[58]



34. ábra: formázott szöveg

35. ábra: generált[59] *szófelhő*

Bár a szöveg (pontosabban annak magyar fordítása) adott, az interpretációnak számos eszköze van, amellyel jelentős többletinformációt szolgáltatunk. A vers címe a képi megjelenésre utal, de ez a szöveges tartalomban csak a látvány és a közlő belső világa közötti kapcsolataként jelenik meg.

A digitális írástudás nem pusztán a szöveg bevitele, hanem a többlet információnak a szándéknak megfelelő, minél pontosabb, hatékony hozzáadása. A digitális írás lényege, hogy a dokumentum értelmezése könnyű legyen és az eredmény a közlés szándékának megfeleljen. [129]

A témakör oktatása/tanulása során megvizsgálandó, hogy az adott médiaelemek a képi, szöveges és hang *adatok rögzítés*ére milyen *absztrakció*t használnak, az adott absztrakció milyen *tulajdonság*okkal rendelkezik, mennyire felel meg a közlési szándéknak és hogyan segíti az

[58] A képvers és szöveg forrása: https://hu.wikisource.org/wiki/A_megsebzett_galamb_és_a_szökőkút [2021.10.30]
[59] Készült a szöveg felhasználásával https://wordart.com/ [2021.10.30] használatával.

*értelmezés*t. A NAT tervezet digitális írástudás fejezetének fejlesztendő területei korosztálytól függően csoportosítják a médiákat, az eredménycélok jellemzően az adott dokumentumtípusok alkalmazási ismeretére vonatkoznak, de a terjedelmi korlátozások miatt a területek kifejtése nem nevezhető strukturáltnak, az ezek alá rendelt eredménycélok néha csak utalást tartalmaznak a sokféle ismeret elvárására.

- *Rajzos dokumentumok elektronikus létrehozása*
- *Adatok értelmezése, csoportosítása, tárolása*
- *Dokumentum létrehozása szövegszerkesztő és bemutatókészítő alkalmazással*
- *Különböző típusú dokumentumok iskolai, tanórai, hétköznapi célú felhasználása*
- *Multimédiás elemek készítése*
- *Nagyméretű szöveges dokumentumok szerkesztését elősegítő eszközök*
- *Multimédia és webes dokumentumok szerkesztése és készítése*
- *Grafikus ábrák készítése és képszerkesztés*
- *A probléma megoldásához szükséges módszerek és eszközök kiválasztása*

Több fejlesztendő terület említi a multimédiát, ahol a „multi” a kép, hang, videó kombinációt jelenti. Fentebb láthattuk, hogy a szöveg (mint a gondolat nyelvi kifejezése) digitális megjelenése számos formátumban lehetséges, így a multimédia minden változatában implementálható. Másrészt, az eredménycélok között többször olvasható „szövegszerkesztő” alkalmazást a multimédiás alkalmazásoktól elkülönítve találjuk, de benne a szöveg mellett képi elemek megvalósítását is elvárja. Funkcionálisan a tárgyalt szövegszerkesztő is multimédiaalkalmazás, ugyanakkor célszerű a kép, hang, videó mellett külön médiának vagy médiaelemnek tekinteni a szöveget. A fogalomzavart fokozza, hogy a multimédiás dokumentumok mellett külön említik a webes dokumentumot.

A digitális írástudás témakör látványosan megáll az alkalmazástípusok tanítása szintjén. Ezért kérdéses, hogy miért kellene informatikából egyik vagy másik alkalmazást tanítani, ami az informatikaoktatás devalválódását eredményezi. A digitális írástudás a digitális eszközök – alkalmazások, appok – *hatékony felhasználás*a érdekében szükséges, ehhez *informatikai gondolkodás*ra van szükség. A digitális írástudás programok *célszerű használat*át jelenti. A programok az informatika tudomány termékei, ebből következően a működésüket informatikai szempontok figyelembevételével kell vizsgálni. A digitális írástudás témakörben feltüntetett alkalmazástípusok felhasználói ismeretén túl szükséges az alkalmazások működésének a modellezése és a modellből következő felhasználói elvek ismerete.

Az aktuálisan tanított alkalmazástípusok felsorolásán túllépve, a témakör informatikai tartalmához az egyes médiákat külön kell jellemezni és értelmezni kell a különböző multimédiás dokumentumokon belül a szerepüket. Az informatikai megközelítést az *adatok*kal és *modellek*kel operáló *absztrakció* jelenti.

Médiaelemek

A grafikai, képi elem két fajtáját különböztetjük meg. Az egyik a rasztergrafikus (pixelgrafikus), melynek elemi objektuma a pixel. Ennek tulajdonságai a színkomponensek értéke. A kép pixelek fix méretű táblázata, az adatok időfüggetlen 2D megjelenítésére alkalmas. Előállítása programmal vagy szenzorokkal kapott adatokkal (pl. fényképezés) lehetséges. A másik módja a kép absztrakciójának a vektorgrafikus ábrázolás. Ebben az esetben az elemi objektum az alakzat, amelynek tulajdonsága a határát alkotó pontok és ezeket összekötő vonalak listája, a vonalak és a belső terület szín/mintázat tulajdonsága, mérete…. A vektorgrafikus kép az alakzatok egymáshoz képest rögzített helyzetű összessége. A képek tárolása jellemzően binárisan történik, de van karakteres forma is (svg). A képre jellemző a tároláshoz alkalmazott tömörítés, ami sok formátum esetén veszteséges. Az informatikatudás része kell legyen a tömörítési eljárások vázlatos ismerete, hatása a kép minőségére. Vektorgrafikus képnek tekinthetjük a diagramokat, szófelhőt is, ahol a kép előállításához a megjelenítést befolyásoló adatokból program generálja az alakzatokat.

A 3D elemeknek többféle absztrakciója és megjelenítése létezik. A legegyszerűbb 3D-ábrázolás valójában kétdimenziós, valamilyen perspektivikus ábrázolást használ a megjelenítésre. Két 2D képből színszűrővel és megfelelő eszközzel 3D érzését lehet kelteni. A 3D tervezőprogramok térbeli vektorgrafikus absztrakciót használnak, de a megjelenítés képernyőre vagy nyomtatásra jellemzően kétdimenziós projekció. A 3D nyomtatás ezeket a térbeli testeket képes előállítani, de kérdéses, hogy a létrejövő tárgy média elem-e. Inkább végtermék. (Azt gondolom, hogy egy igazi 3D digitális elem hologramként jelenik meg, körbejárható, és drónhoz hasonló szerkezettel („repülő egérrel”, azaz denevérrel) szerkeszthető.)

Hang digitalizálása során a térbeli tulajdonságokat csatornákra bontással lehet tárolni. A hang absztrakciójában a hanghullám tulajdonságainak időbeli állapotváltozásai dominálnak. Kérdés, hogy a csatornák számossága jelent-e dimenziót. A csatornák a 3D anaglif megjelenítéshez hasonló elven a dekódoló egységek megfelelő elhelyezésével teszik lehetővé a térbeli érzékelést, ezért a sztereó érzékelés inkább több médiaelem egyidejű befogadása, mintsem a hang térbeli absztrakciója.

Szöveg médiaelem legkisebb objektuma a karakter. A karakter az adott nyelvre jellemző legfeljebb 4 bájtos, tipikusan 1 bájtos adat, a szöveget a karakterkészlet egydimenziós sorozataként tároljuk. A nyelvi egységeket (jellemzően szavakat) speciális karakterrel, szóközökkel választjuk el egymástól, a nagyobb gondolatok határolókaraktere a bekezdés vége jel. A beszédben hallható egyéb nyelvtani tagolásokat mondathatároló írásjelekkel, központozással, kötőjelekkel jelöljük. A szöveg megjelenése lehet hang (felolvasó eszközzel) vagy 2D kép, ekkor – jellemzően – a szöveg sorokra tördelve olvasható. A szövegelem megjelenését hozzáadott tipográfiai tulajdonságokkal adhatjuk meg. Ezzel együtt a szöveg nagyobb egységei objektumként értelmezhetők, melyek megjelenéssel kapcsolatos tulajdonságokkal rendelkeznek. A formázott szöveg tárolása lehet bináris (doc), de napjainkban sokkal jellemzőbb a tipográfiai jellemzők jelölőnyelven történő megadása, az adatok és formátum szöveges vagy veszteségmentes tömörítéssel történő tárolása.

Az animáció határesete a multimédiának, hiszen jellemzően képek, grafikák egymás után történő megjelenítéséről van szó. Hasonlóan a felvett videó is a képek (framek) időbeli egymásutániságát megtartó sorozata. Mindkettő tekinthető a 2D – esetleg 3D – ábrázolás időbeli kiterjesztésének, amely azonos médiaelemek sorozata. Azonban az időbeli kiterjedés szinte természetes velejárója, hogy hang is adható a képi médiához, ezért inkább multimédiákhoz sorolható.

Multimédia alkalmazások

A „multimédia” fogalomnak – ahogy az sok más informatikai fogalomra is igaz – több értelmezése volt, van és talán még lesznek újabbak is. Informatikaoktatás szempontjából azonban lényeges, hogy multimédiás tartalom előállításához először az egyes médiaelemek jellemzőit, adatmodelljét célszerű megismerni, ezt követően lehet egy-egy cél érdekében ezen elemekből többfélét kombinálni. A multimédiás alkalmazások célszerű felhasználásához ismerni kell az egyes elemek objektummodelljét és azt is, hogy ezen elemeket a multimédiás alkalmazás hogyan kombinálja. Ezért tartom fontosnak például a Paint mint egyszerű, tisztán rasztergrafikuskép készítő alkalmazás tanítását. Ezen az alkalmazáson keresztül elemeztem, hogy hogyan lehet a menü és az alkalmazás gyakorlati trükkjei mellett a rasztergrafikuskép adatmodelljét is tanítani, az informatikai ismeretek mentén akár a kép programozással történő módosítását is bemutatni. [130] A multimédiás alkalmazások tanítása előtt vagy annak kezdeti szakaszában kell a lehető legtisztább formában megismerkedni a többi felhasználni kívánt médiaelemmel is, így a karakterek kódolásával, nyelvtani és helyesírási szabályokkal is.

Bár a témaköröknél külön említésre kerül, multimédiás alkalmazások a mai fejlett szövegszerkesztők és a weblapkészítő eszközök is. Érdemes megfigyelni az alapértelmezett dokumentumaik néhány hasonlóságát és eltérését. Mindkettőnek az alapja formázott szöveges dokumentum; a főbb objektumok: karakter, bekezdés, szakasz. Mindkettőben van mód az adatok strukturált elrendezésére, használhatunk táblázatot. A szövegszerkesztővel konstans szélességű és magasságú lapokra tervezzük a dokumentumot, míg weblap készítéskor a dokumentum szélessége a felhasználói oldal által meghatározott paraméter és a lapnak nincs előre megadott hossza. A szövegszerkesztővel készített dokumentum a képeket beágyazva tartalmazza, a weblapon csatolással, azaz külső forrásra hivatkozással adjuk meg a többi médiát. Ráadásul a weblapba videó és hang csatolása is értelmes, míg ezek a szövegszerkesztők esetén, nyomtatásra készülve nem relevánsak. Szövegszerkesztőben az egyes formázási stílusoknak van neve, a stílusok egymásból származtathatók, amit módosít a konkrét lokális formázás. Weblapkészítésben a css jellemzően a meglévő elemekhez definiálja a megjelenést, amely a globálistól a lokális felé, valamint olvasási sorrendben kiértékelve felülírva alakítja ki a megjelenő képet. Informatikai szempontból a kétféle alkalmazással lényegében ugyanazt a tartalmat azonos alapelven, de az eltérő cél miatt különböző megoldásokkal kódoljuk a dokumentumokba.

Az inkább képi megjelenésre optimalizált, médiaelemeket mint objektumokat strukturáltan tartalmazó infografika-készítő és a bemutató-készítő alkalmazások között az időbeliség kezelése a legnagyobb eltérés, ugyanakkor filozófiai – és ezért felhasználási területben is – különbség van a diákra (slide-okra) tervezett bemutató-készítő és a Prezi gondolattérkép jellegű megoldásai között. Animációt és videót bemutató-készítő alkalmazásokkal is lehet készíteni, csak a kézi vezérlés helyett időzítést kell alkalmazni.

Az animációkészítés oktatásával előkészítjük a programozást is. Az objektumok megjelenésének vezérlése, mozgatása, átalakítása lényegében deklaratív programozás, eljárások paraméterezett meghívását jelenti. A blokknyelven programozás csak annyiban ad ennél többet, hogy ott az eljárást vezérlési elemekből kell elkészíteni, míg a bemutatókészítő programokban előregyártott algoritmusokból választhatunk. Emiatt az informatikaoktatásban sokkal nagyobb a szerepe a prezentációkészítésnek, mint a későbbi felhasználási területeken. Az értekezleteken vetített bemutatók nem biztos, hogy növelik a hatékonyságot, mert jellemzően az a szerepe, hogy az előadó gondolkodási sorrendjét rákényszerítse a hallgatókra. De a tanítás/tanulás folyamatban fontos szerepe lehet abban, hogy a diák átgondolja egy folyamat szekvenciális és párhuzamos lépéseit. A szöveg megjelenésének animálása is kifejezhet belső (programozói) szándékot,

de ez viszonylag ritka. A prezentációk jelentős részében a teljes dokumentumra jellemző megjelenítési stílust formázzuk az animációval. Az informatikai gondolkodás fejlesztése szempontjából a „történetmesélés”, a gondolatmenet kifejtése fontosabb.

Példák a gondolatmenet prezentálására, algoritmizálására:

- Egy-egy kommunikációs jelenetet bemutató prezentáció – a szereplők helyzetváltozásával, a szövegbuborékok megfelelő sorrendű megjelenítésével – felér egy hétköznapi algoritmus leírásával.
- A tanult algoritmusok (például: összeadás, szorzás) lépésenkénti bemutatása, megjelenítése az elkészítés során tanít, bemutatáskor a tanár ellenőrizheti a helyes műveletvégzést.
- Egy szélkerék forgatásában az elemek csoportba foglalt egységként kezelése; egy hétszegmenses kijelzőn visszaszámlálás megjelenítése az objektum részeinek egyedi párhuzamos kezelése megmutatja az „egyben”, illetve „egyidőben” végrehajtott művelet közötti különbséget, fejleszti az objektum szemléletet, a párhuzamos és szekvenciális vezérlést. Ebben a két feladatban megjelenik az időzítés, az adott idejű hatás, a végtelenítés és a következő eseményig tartó lejátszás fogalma is.
- Az animációk választása a bemutatott fogalom ismeretét is mutatják. A kislány nem csak feltartja a kezét, hanem integet; a tűz lobog; a háromszög súlyvonala nem bepörög, hanem a csúcsból a szemközti oldal felezőpontja felé megjelenik; a hal beúszik (fejével előrefelé); a hópihe, a falevél a levegőben táncolva halad lefelé…

A programozás előkészítéseként érdemes megfogalmazni, hogy az egyes animációk az *objektum*ok mely *tulajdonság*ait hogyan módosítják, de informatikai és digitális írástudás szempontjából fontosabb, hogy a gondolathoz leginkább illeszkedő kész *eljárás*t ki tudja választani a diák. Ez a gondolat kódolása, az *algoritmizálás*.

Személyes példa

2002-ben, az ELTE-n Informatika a matematikaoktatásban címmel tartottam gyakorlatot. Itt vettem észre, hogy a leendő matematikatanárok a később tanítandó tételek bizonyítását nem jól prezentálják. A megtanult bizonyítások jellemzője, hogy a megfogalmazásban először állít, majd indokol. Például: A két háromszög hasonló, mert két-két szögük egyenlő. Az animációban először kellene megmutatni, hogy a két-két szög egyenlő, majd mozgatással, nagyítással megmutatni a hasonlóságot. Később, a 9–11. évfolyamon, a prezentáció oktatásába mindig bevettem valamely geometria tétel bizonyítását animációval.

Mindig egyéni projektként, tetszőleges forrásfelhasználással. A beadott munkákból egyértelműen látszott, melyek a meg nem értett összefüggések. A legdurvább hibák téves jelölések, rossz helyre berajzolt vonalak formájában láthatók (ez a matematika dolgozatokból is ismert), de alapvető hiányosság jele a hibás megjelenési sorrend vagy hibás animációtípus alkalmazása. Ezekben az esetekben a jól bemagolt, de nem megértett bizonyítás érhető tetten. A legtöbb diák az 1–3 továbbfejlesztési, javítási lehetőség során rendezi logikai sorba a bizonyítás lépéseit. Ez a folyamat nem más, mint a gondolatsor algoritmikus kifejtése. Ezután már nem csak felmondani tudja a bizonyítást, hanem érti is, amit mond, hiszen egy gépnek is el tudta magyarázni.

A multimédiás tartalmat érdemes a vétel (dekódolás) alapján is informatikai szempontok alapján megvizsgálni. Melyek azok, amelyek a lineáris adatsor közvetítésére alkalmasak (pl.: hang), melyek engednek a feldolgozásra több bejárási utat (pl.: weblap) és melyik médiák adnak a feldolgozás, a dekódolás oldalán szabadságot (pl.: kép esetén a szemlélő fókuszál egy-egy részletre). A médiákat az ember dekódolja. Ennek megfelelően már kódoláskor figyelemmel kell lenni a dekódoló képességeire. A vizuális tartalmak dekódolása sokkal gyorsabb, mint az audióé, kivéve, ha gyengénlátó, esetleg vak az illető. A hang egyedi dekódolása (füldugóval) a közvetlen környezet érzékelését gyengíti, a kakofóniából a szükséges hangok kiszűrése fárasztó és nehézkes. A 3D ábrázolás realisztikusabb, de – főleg mozgó változata – az embert érő vizuális és mozgásérzékelési ingerek ellentmondásos jelzései miatt rosszullétet okozhatnak (tengeri betegség).

A multimédiás tartalmak készítésekor a *hatékonyság*ot több szempontból kell vizsgálni: Az elkészítés hatékonysága a közlési szándék és az elkészítési idő függvényében értelmezhető. A dokumentum tárolásának hatékonysága méretbeli és adatbiztonsági kérdéseket vet fel. A felhasználás hatékonysága a közvetített információ megszerzésének lehetősége, ideje, teljessége alapján jellemezhető.

Bár a témakörön belül nincs nevesítve, a táblázatkezelők is multimédiás alkalmazások; az adatok elrendezett megjelenítése, formázottsága és a diagramok tipikus médiaelemek, az infografikának is részei lehetnek. Az adatbáziskezeléshez rendelt felületek, az űrlapok és jelentések szintén multimédiás dokumentumok. E két alkalmazástípust az teszi különlegessé, hogy a megjelenő szöveges és képi elemek hátterében értékként kezelt adatokkal, a megjelenítést befolyásoló programok, függvények, eljárások működnek.

A digitális írástudás témakör eredménycéljai között szerepel a tartalomkezelő alkalmazás használata is, ugyanakkor számos már ma is használt multimédia alkalmazás és médiaelem típus nincs nevesítve. Nem szerepel digitális jegyzettömb (például OneNote) ismerete és használata. Tekinthető a digitális jegyzettömb is tartalomkezelő rendszernek? Egy kicsit más, inkább egy draft verziója a tartalomkezelőnek. Milyen médiaelem az egyenlet? Az egyenletszerkesztők szöveges alkalmazások? Valószínűleg annak tekintendők, de azért elég különlegesek, főképp, ha matematikai értelmező is tartozik hozzá és például diagramot képes generálni egy függvény-hozzárendelésből. Utolsó példaként, mi a helyzet a kiadványszerkesztő eszközökkel? A tudományos kiadványok LaTeX nyelvű szerkesztői szövegszerkesztők lennének? A kiadványkészítő Publisher a szöveget dobozokban helyezi el, ezzel keverednek benne egy bemutató-készítő és egy szövegszerkesztő tulajdonságai.

A digitális írástudás nem csak azt jelenti, hogy ismerünk és célszerűen fel tudunk használni egy tucatnyi alkalmazást gondolataink, adataink közlésére, hanem azt is, hogy ezeknek az alkalmazásoknak ismerjük a *működési elv*ét: az *adat*ait (*objektum*ait) és *algoritmus*ait. Értjük az alkalmazások működése eltéréseinek az okát és ennek ismeretében *választ*juk meg a közlés, tárolás és feldolgozás szempontjából legmegfelelőbb alkalmazást, multimédia elemeket és *megoldási módszerek*et. A digitális írástudás képessé tesz arra, hogy új alkalmazások megjelenésekor annak működési elvét – a korábban megismertekkel összehasonlítva – *önállóan* felderítsük, *hatékony* használatát elsajátítsuk.

6.2.3

IX/4. Problémamegoldás informatikai eszközökkel és módszerekkel (3.)

A téma címe alapján a tantárgyon belül itt oktatunk informatikát. Valójában a témakörön belül az ember-számítógép kommunikáción van a hangsúly. Középpontban a gép vezérlése, a gépen tárolt adatok feldolgozása áll. Informatikatudományra fókuszálva, a programozás különböző módszereinek az alapjai szerepelnek ebben a témában: imperatívan programozás, deklaratívan adatbázis-kezelés, az objektum orientált és esemény-vezérelt programozáshoz leginkább a robotika nyújt több-kevesebb alapot. A táblázatkezelők használata a funkcionális programozás gondolkodásmódját igényli, adatkezelése, megoldásai a programozást több szempontból modellezik. Ezért a programozással kapcsolatos ismeretek a táblázatkezelés oktatása során előkészíthetők. Erre mutattam néhány példát a How to Teach Programming Indirectly – Using Spreadsheet Application [133] cikkemben. Az oktatási gyakorlatban használható példák elméleti rendszerbe foglalásával bemutatható, hogy a táblázatkezelés a programozás általános elő-

készítő alkalmazása lehet, ahogy ezt a Using the Spreadsheet Paradigm to Introduce Fundamental Concepts of Programming to Novices [134] cikkben olvashatjuk. Bár eszerint az elemzés szerint a ciklus megvalósítása nincs meg a táblázatkezelőkben, a képletek másolása egész közel van a ciklusképzéshez. A tömbfüggvények használatával pedig egy képletbe is foglalhatjuk a műveletek ciklikus elvégzését. A SPREGO [135] gondolkodási módszer a táblázatkezelő eszközeit programozási nyelvként használja, így képez kapcsolatot a táblázatkezelés és programozás oktatás között. A láthatóan szoros kapcsolat miatt a programozást előkészítő blokkprogramozás és a robotika mellett az egyik legáltalánosabb irodai alkalmazásnak, a táblázatkezelésnek is helye van a problémamegoldás eszközei között.

Az adatbáziskezelő alkalmazások a táblázatkezelőknél is jobban hasonlítanak a programok fejlesztő környezetéhez. Két nyelv, az adatdefiníciós és a lekérdezőnyelv kódjai futnak a kattintgatásokkal vagy szövegesen kiadott utasítások hátterében. Bár az eredménycél minimumok között nem szerepel az SQL nyelv ismerete, a feladatokat lekérdező-rácson (QBE felületen) kiadva, a blokknyelvekhez hasonlóan programozásról beszélhetünk.

Az alábbiakban a fejlesztendő területek alatt az egyes eredménycélok kulcs kifejezéseit gyűjtöttem ki. A táblázat- és adatbáziskezeléssel kapcsolatos eredménycélok egy része funkcióját tekintve a digitális írástudáshoz tartozik (ezeket kihagytam). Ami ezen felül olvasható, az szoros kapcsolatban van az algoritmizálás, programozás témákkal, sőt, a „programozás” fogalom értelmezésétől függően, tekinthetjük deklaratív nyelven történő programozásnak.

Alsótagozaton

- *A probléma megoldásához szükséges módszerek és eszközök kiválasztása*
  - *…értelmez néhány egyszerű problémát, megoldási lehetőségeket játszik el, fogalmaz meg, valósít meg … információkat keres és használ fel …*
- *Algoritmusok vizsgálata, előállítása*
  - *Felismer, eljátszik, végrehajt … elemi lépésekből álló, adott sorrendben végrehajtandó cselekvést; … algoritmust elemi lépésekre bont, értelmezi a lépések sorrendjét, megfogalmazza az algoritmus várható kimenetelét …*
- *Kódolás, folyamatok irányítása, a robotika alapjai*
  - *Az eszköz mozgását értékeli, hiba esetén módosítja a kódsorozatot … kódsorozatot tervez és hajtat végre, történeteket, meserészleteket jelenít meg az eszközzel, például padlórobottal.*

Felsőtagozaton

- *Egyszerű algoritmusok elemzése, készítése*
  - *Értelmezni képes az algoritmus végrehajtásához szükséges adatok és az eredmények kapcsolatát; megkülönbözteti, kezeli és használja az elemi adatokat.*
- *A kódolás eszközeinek ismerete, a blokkprogramozás építőelemeinek használata*
  - *Probléma megoldásához vezérlési szerkezetet alkalmaz blokkalapú programozási nyelven.*
  - *Szereplőközpontúan mozgásokat vezérel virtuálisan (képernyőn) és valóságban (például robottal).*
- *Adatok kezelése (Táblázatkezelés)*
  - *Cellahivatkozásokat, matematikai tudásának megfelelő képleteket, egyszerű statisztikai függvényeket használ.*
- *Tantárgyi problémák vizsgálata digitális eszközökkel*
  - *... problémákat old meg táblázatkezelő program segítségével. ... hétköznapi jelenségek számítógépes szimulációja ... a szabályozó eszközök hatásai.*

Középiskola

- ***Algoritmizálás**, módszerek, eszközök használata, típusalgoritmusok*
  - *... elemi adattípusok ...: egész, valós szám, karakter, szöveg, logikai; ... egy algoritmus-leíró eszköz ... típusalgoritmus…*
- ***Programozási nyelv** fejlesztői környezete, vezérlési szerkezetek*
  - *... programozási nyelv fejlesztői környezetének alapszolgáltatásai; ... szekvencia, elágazás és ciklus segítségével egyszerű algoritmust létrehozni, és azt egy formális programozási nyelv segítségével megvalósítani; ... feladat megoldásainak helyességét tesztelni.*
- *Adatkezelés **táblázatkezelő** alkalmazással*
  - *... táblázatkezelővel adatelemzést és számításokat végez; ... függvényeket célszerűen használ ... nagy adathalmazokat kezel; az adatokat diagram segítségével szemlélteti.*
- *Adatkezelés **adatbáziskezelő** rendszerrel*

- *… adatbázis-kezelés alapfogalmai; keres, rendez és szűr; adatokat visz be, módosít és töröl; űrlapok…, jelentések…*

– *Számítógépes szimuláció*
  - *… használja a hétköznapi, oktatásához készült szimulációs programokat; …kezdőértékek változtatásának hatásai…*

Az informatikaoktatás tartalma szempontjából azt érdemes megvizsgálni, hogy a témakörön belül az egyes programozással kapcsolatos fogalmak hogyan jelennek meg, hogyan fejlődnek. A korábban tárgyalt témakörökhöz képest itt sokkal hangsúlyosabban jelenik meg, hogy valamilyen formában mindig programozást, az ehhez szükséges fogalmakat tanítjuk vagy fejlesztjük a programozáshoz szükséges készségeket.

Adat

Az adat absztrakciója az informatika tanulásának kezdetén elég elnagyolt, lényegében médiaelem. Virtuális entitásokat használ a gyerek. Ezek részletesebb megismerése, jellemzése, leírása során alakul ki az adattípus, illetve az objektum fogalma. A robotika alapjainak tanulása során eszközöknek egyes tulajdonságait kell beállítani, egy véges listából választva. A táblázatkezelő kezdetben csak az adatok elrendezését, strukturálását segíti, de már az első pillanatban érdemes tisztázni a szöveg és a szám megjelenésének eltérését, azt, hogy a karaktersorozatként beírt adat átalakulhat, ilyenkor nem lóg túl a cellahatáron, és jobbra igazított lesz. Később tisztázni kell, hogy az ″123″ szöveg, a 2020. február 2. viszont szám jellegű, értékkel rendelkező adat. A táblázatkezelők használatához szükséges ismeret, hogy a számítógép digitálisan működik és az adatokat alkotó jeleket binárisan tárolja. Ismerni kell a kettes és tízes számrendszerek közötti váltást és azt is, hogy a számokat nem tizedestört, hanem kettedes tört alakban értelmezik a gépek. Bőséges tapasztalattal kell rendelkezni arról, hogy a táblázatkezelő adatértelmezője gyengén típusos, az automatikus adatkonverziót megadott formátumokra illeszkedés alapján végzi.

Az első probléma a tizedespont dátumelválasztóként értelmezése szokott lenni, de alapismeret lenne az is, hogy a százalékjel százzal osztja az eléje írt számot, a ’Ft’ viszont csak megjelenített mértékegység, nem társul hozzá árfolyam (50% értéke 0,5; a 10 Ft, illetve 20 € tartalmú – és formátumú – cellák összege szoftvertől és a tagok sorrendjétől függően 30 Ft, 30 € vagy 30 is lehet). Táblázatkezelésoktatás részeként értelmezzük a szöveg, a szám (érték), a logikai, a hivatkozás és a tömb adattípust, az adatbáziskezelés programozási nyelvei típusosak, ami a fogalmak további pontosítását és tudatos alkalmazását tennék lehetővé.

A robotika lehetőséget ad arra, hogy az emberi adatbevitel helyett szenzorokról származó inputtal, és az eszközök állapotváltozását eredményező outputtal dolgozzon a tanuló. Ez a fizikai kapcsolat segíti a virtuálisan létrehozott objektum absztrakcióját. A legegyszerűbben érzékelhető inputok (kapcsolók) és vezérelhető outputok (LED-ek) programozásával a processzor működési elvének megértéséhez is közelebb juthat a tanuló. Ezért a robotika témakörben nem csak a számítógéppel (táv)vezérelt robotok működését kell tanulmányozni, hanem a közvetlen, mikrokontrolleres vezérlést, a beágyazott rendszerek működését is, illetve a processzor működésének absztrakciójához a digitális technika alapjait, a logikai kapuk működését is célszerű tanulmányozni.

A programozás tanításakor az ebben és a többi témakörben korábban szerzett tapasztalatokra lehet alapozni a változó, a különböző adattípusok és az osztály, illetve az objektum fogalmát, amelyet a programozási nyelv tanításakor tudunk kreatívan használhatóvá tenni: adatot tudatosan létrehozni, módosítani, megszüntetni. A diáknak az adat több absztrakciós szintet kell megértenie.

Példák az adattal és objektummal kapcsolatos fogalmak kialakításához alkalmas témákra

- Rasztergrafika: pixel, táblázat, kép; a pixel objektum tulajdonságai
- Vektorgrafika: alakzatok, alakzatok tulajdonságai, alakzatok átalakítása
- Szöveg, dokumentum objektumai: karakter, bekezdés, szakasz; teljes dokumentum
- Táblázat, tábla, sor, oszlop, cella, rekord, mező, adat, hivatkozás, kulcs (kapcsolat)
- Szereplő, karakter, profil, sprite, metódus, függvény

Amellett (vagy ahelyett), hogy például a „digitális írástudás” keretei között tanítjuk az alkalmazások használatát, az alkalmazások informatikai vonatkozásait kellene tanítani. Ezen belül az adat, az objektum és hozzá kapcsolódó fogalmakat is vizsgáljuk: aggregáció, kompozíció, asszociáció, tulajdonság, művelet, eljárás függvény; tulajdonságok alapértelmezett értéke (default beállítások); objektum (adat) létrehozása, paraméterezése, törlése, tulajdonságainak módosítása és módosulása, másolása, közvetlen és közvetett elérése.

A múlt században születettek számára az egyszerű adat, a karakter, az egész szám volt a tanulás kiindulási pontja, ebből építette fel a szakember az objektumokat. Az információs társadalom generációi objektumokon keresztül ismerik meg a digitális világot, ezért az oktatás során az objektumok elemző (informatikai) megismerése a kiindulási pont, innen kell – kezdetben szakmai megnevezések nélkül – a különböző elemi és összetett adatok fogalmait megismerniük.

Elemi adatok

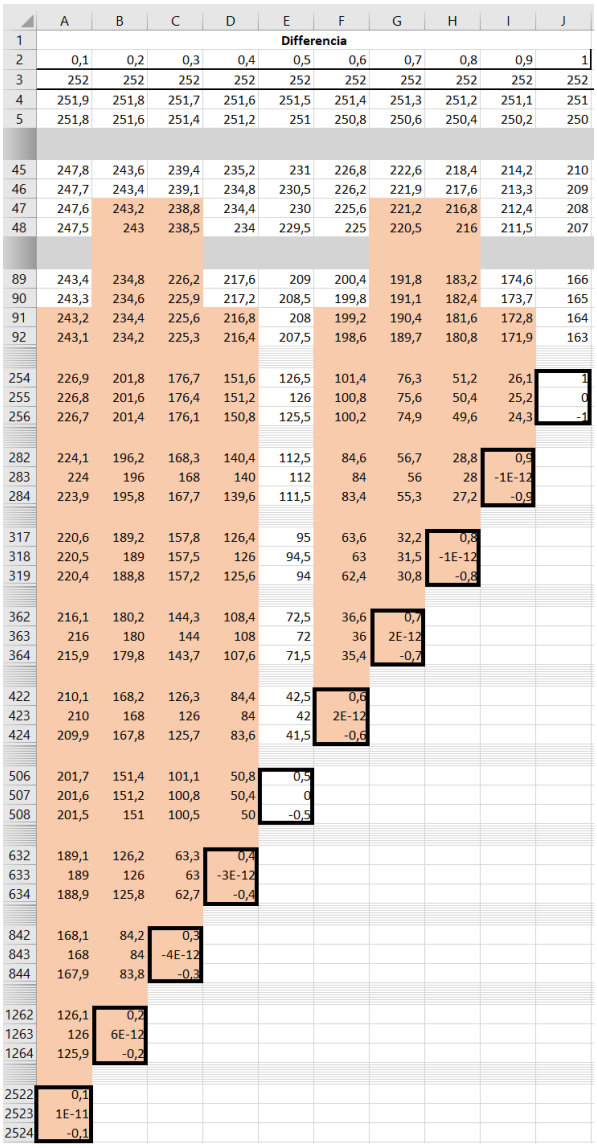
Az informatikatudomány újraértelmező természete itt is megmutatja magát. Az objektumorientált nyelvekben az integer is objektum. Minden objektum. De, hogyan értelmezhető az elemi adat? Az elemi adat tárolása csak egy konkrét érték tárolását jelenti, amelyre a műveletek gépikódban, processzorművelet szintjén definiáltak. A processzor elemi adatokkal dolgozik, az ALU az elemi adatokon tud műveleteket végezni. Ezért az adat fogalmának megértéséhez szükséges a robotika és ezen keresztül a digitális technika alapjainak is az oktatása. Ismeret szintjén meg kell teremteni a kettes számrendszer, a bitműveletek, az aritmetikai és logikai műveletek és a fizikai eszközök (szenzorok, inputok és outputok) absztrakciók kapcsolatát.

Amikor matematikából a diák elkezd a folytonosság és végtelen fogálmával ismerkedni, ideje alaposabban megismernie informatikából a digitalizálás jellemzőit. A matematikai kerekítés szabályaival egyidőben kell megmutatni a mintavételezés és kvantálás tulajdonságait. Matematikaoktatásban a kerekítés oka a végeredmény nagyságrendi megjelenítése, informatikából az adat létrehozásának, vagy mérésnek az eredménye, a valósághoz közelítő érték. Miközben matematikában – formális jelölésekkel – pontos számítások elvégzését tanulja a diák, meg kell mutatni, hogy a digitális világban mely műveletek eredménye pontos, a kvantálás hogyan befolyásolja az adatokkal végzett számítások eredményét. Tisztázni kell a matematikában tanult számhalmazok és az informatikában használt adattípusok közötti kapcsolatokat és (főleg) különbségeket. Példákon keresztül meg kell mutatni a digitális adatkezelés használhatóságának okát: Ismert pontatlanságú, ellenőrzötten értékálló adatok jobban használhatók, mint az elvileg végtelenül pontos, de ellenőrizhetetlen, összehasonlíthatatlan eredmények.

Példák digitális tárolás értelmezésére:

- Számtani sorozat rekurzív számítása során a digitális tárolás kerekítési hibája összeadódik. Alkalmazói programtól, fordítóprogramtól függ, hogy ez a kerekítés mikor lesz kimutatható. A 36. ábra azok a cellák vannak feltételes formázással kiemelve, amelyekben a cellaérték nem egyezik meg az explicit módon megadott értékkel ($a_n = a_0 – n \cdot d$).
- Hasonlóan, a 36. ábra sorozatainál megvizsgálható, hogy hányadik lépésben éri el a sorozat a 0-t. A választott kiindulási értékkel (252), differenciaként minden tizedre a [0,1; 1] tartományból a 0 érték is a sorozat tagja. Látható – programot írva rá tapasztalható –, hogy a lépésenkénti csökkentés nulla helyett sokszor a kerekítési hibát írja ki. A nem egész (azaz lebegőpontos, valós) számábrázolás esetén a tárolt jegyek

száma véges. A kerekítési hiba az eredmény nagyságrendjének csökkenésével valódi értéknek látszik.



	A	B	C	D	E	F	G	H	I	J
1	Differencia									
2	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1
3	252	252	252	252	252	252	252	252	252	252
4	251,9	251,8	251,7	251,6	251,5	251,4	251,3	251,2	251,1	251
5	251,8	251,6	251,4	251,2	251	250,8	250,6	250,4	250,2	250
45	247,8	243,6	239,4	235,2	231	226,8	222,6	218,4	214,2	210
46	247,7	243,4	239,1	234,8	230,5	226,2	221,9	217,6	213,3	209
47	247,6	243,2	238,8	234,4	230	225,6	221,2	216,8	212,4	208
48	247,5	243	238,5	234	229,5	225	220,5	216	211,5	207
89	243,4	234,8	226,2	217,6	209	200,4	191,8	183,2	174,6	166
90	243,3	234,6	225,9	217,2	208,5	199,8	191,1	182,4	173,7	165
91	243,2	234,4	225,6	216,8	208	199,2	190,4	181,6	172,8	164
92	243,1	234,2	225,3	216,4	207,5	198,6	189,7	180,8	171,9	163
254	226,9	201,8	176,7	151,6	126,5	101,4	76,3	51,2	26,1	1
255	226,8	201,6	176,4	151,2	126	100,8	75,6	50,4	25,2	0
256	226,7	201,4	176,1	150,8	125,5	100,2	74,9	49,6	24,3	-1
282	224,1	196,2	168,3	140,4	112,5	84,6	56,7	28,8	0,9	
283	224	196	168	140	112	84	56	28	-1E-12	
284	223,9	195,8	167,7	139,6	111,5	83,4	55,3	27,2	-0,9	
317	220,6	189,2	157,8	126,4	95	63,6	32,2	0,8		
318	220,5	189	157,5	126	94,5	63	31,5	-1E-12		
319	220,4	188,8	157,2	125,6	94	62,4	30,8	-0,8		
362	216,1	180,2	144,3	108,4	72,5	36,6	0,7			
363	216	180	144	108	72	36	2E-12			
364	215,9	179,8	143,7	107,6	71,5	35,4	-0,7			
422	210,1	168,2	126,3	84,4	42,5	0,6				
423	210	168	126	84	42	2E-12				
424	209,9	167,8	125,7	83,6	41,5	-0,6				
506	201,7	151,4	101,1	50,8	0,5					
507	201,6	151,2	100,8	50,4	0					
508	201,5	151	100,5	50	-0,5					
632	189,1	126,2	63,3	0,4						
633	189	126	63	-3E-12						
634	188,9	125,8	62,7	-0,4						
842	168,1	84,2	0,3							
843	168	84	-4E-12							
844	167,9	83,8	-0,3							
1262	126,1	0,2								
1263	126	6E-12								
1264	125,9	-0,2								
2522	0,1									
2523	1E-11									
2524	-0,1									

36. ábra:
$a_0 = 252$; $a_n = a_{n-1} - d$; $d = 0{,}1 \cdot e$; $e \in \mathbb{N}$, $e \leq 10$
színes kiemelés: $a_n <> a_0 - n \cdot d$; fekete keret: „zérushely” környéke

- Nem lehet feltétel a sorozatos összeadások hibáinak összegzésekor a tagok egyenlősége. Bármilyen lebegőpontos számok összegzése magában hordozza a kerekítési hibák összeadódását.
- Valós (gyakran racionális) számokkal operáló matematikai azonosságok sem ellenőrizhetők számítógéppel. A 2/GYÖK(2) = GYÖK(2) kifejezés értéke HAMIS épp úgy, mint a SIN(RADIÁN(45)) = GYÖK(2)/2 is. Az egyenlőség igazolása csak fordított irányban történhet: a számítógép által kijelzett egyenlőnek látszó értékekről matematikai úton igazolhatjuk, hogy valóban egyenlőek. Matematikai egyenlőség akkor

is lehetséges, ha a számítások (amelyekben kerekítéseket alkalmaztunk) alapján a két mennyiség nem egyenlő.

- Ha az alkalmazásban a dátum-idő egysége a nap, akkor az időpontokkal történő számolás a fentihez hasonló problémákat jelent. Például 8:00 után 8-szor 0:15 (negyedóra) nem biztos, hogy pontosan 10:00-val lesz egyenlő.

Nem tananyag, de tantárgyi kooperációban érdekes informatikai téma annak vizsgálata, hogy

- a világ digitális-e vagy analóg;
- képesek vagyunk-e analóg mérésre;
- „folytonos" vonallal rajzolt függvényeink minden esetben diszjunkt ponthalmazok;
- egy fénykép nagyításával lehet-e további részleteket megjeleníteni, adatokat kinyerni;
- esetleg: mi a Plank-idő és a Plank-hossz.

Összetett adatok

Visszagondolva tanulmányaimra, a sorozatokat – és ezzel kapcsolatban az indexelést – körülbelül 16 éves koromban tanultam először. Ezután 5 évvel, 21 évesen tanultam programozni BASIC-ben, ahol a számokból vagy karakterekből álló kétdimenziós táblázat volt a legbonyolultabb adatszerkezet. Az első kezelendő összetett adat a szöveg volt, ezt követte az egészek tömbje. 30 évesen, számítástechnikatanári kiegészítő szakon tanultam a struktúrákról és egy icipicit az objektumokról Pascalban. 34 évesen készítettem az első Windows Form alkalmazást, Visual Basicben, újabb 8 év, mire megírtam C#-formon egy önállóan tervezett alkalmazást, de csak 2 évvel később írtam osztálydefiníciót feladatok megoldásában.

Annak, aki lényegében születése pillanatától grafikus képernyőt használ, az első (összetett) adattípus, amivel megismerkedik, az objektum. Ikonokra bök rá, alakzatokat hoz létre, képek tulajdonságait változtatja meg. Ezt használják ki, a padlórobotok és blokknyelvek is. Az objektum fogalmának kibontásával kapjuk, hogy a tulajdonság lehet rész, azaz egy másik objektum, vagy begépelhető adat (számok, betűk). Az autónak van kereke, a keréknek van színe… Az első alkalmazásokban egyedi objektumokat kezel a gyerek, majd néhány objektumot, de egyedileg tartja számon.

30 évestől kb. 42 éves koromig szedtem össze azt a tudást, amit a mai diákoknak 5–12 éves korban kellene tanulniuk az objektumokról. Azóta természetesen változott a környezet is, de a tanítás során figyelembe kell venni, hogy sokkal fiatalabb korban várunk el bizonyos absztrakciókat, a tanítványok másképp tapasztalják meg a körülöttük levő világot, mint a tanáraik.

A tanterv alapján a 12 éves diák – több éves gyakorlattal az objektumok használatában – tanul táblázatkezelést, megtanulja a cellahivatkozást. A pixelgrafikus rajzolással foglalkozva tapasztalatot szerez a koordinátákkal kapcsolatban. Lényegében érti a kétdimenziós táblázat absztrakciót. A sorozatok matematikából 8. évfolyamon tananyag, így 14 évesen már értheti, hogy mit jelent az, hogy $a_7 = 30$. Ez számára a függvényhez kapcsolódik, így csekély köze van ahhoz, hogy a szöveg karaktersorozat, amiben az egyes karaktereket indexükkel érjük el és megvizsgálhatjuk, hogy a ″Hello″[4] == 'o' igaz-e. Az informatikában tömbnek nevezett adott elemszámú, azonos típusú elemekből álló adatsorozat absztrakciójára ezért külön figyelmet kell fordítani. Korábbi ismeretekből, tapasztalatból nem következik, hogy számjegy gombokhoz, kártyapaklihoz vagy akadályok listájához az adott típusú objektumokból álló tömböt először üresen kell létrehozni (mint egy fiókosszekrényt, fióktartó sínekkel), majd ebbe az egyes elemeket – objektumokat is létre kell hozni (el kell készíteni a fiókokat és beletenni) és a megfelelő megjelenéshez az objektumok kezdeti tulajdonságait is meg kell adni (azaz a fiókba bekerülnek az adatok). Könnyebb a tömb absztrakciója, ha a blokknyelven szerzett tapasztalatok helyett a többi tanult alkalmazásból, leginkább táblázatkezelésből veszünk példákat. Azért is fontos, hogy táblázatkezelésben az adattípusokra kitérjünk, mert ott használunk adott méretű, egész számokat tartalmazó tömböt, illetve a szöveget karaktersorozatként, aminek van hossza, meg lehet adni egyes részeit.

A táblázatkezelők alkalmasak a tömb méretezésének bemutatására. Excel 2016 esetén kiszámolható a „teljesen feltöltött" táblázat fájlmérete. Például 256 munkalap minden cellájába 1 karaktert beírva a fájlméret 1 TiB körüli érték lenne. A táblázatkezelők vagy eleve maximalizált sor és oszlopszámot, munkalapszámot és szöveghosszt engednek, vagy a rendelkezésre álló memóriától függ a bővítés engedélyezése. Excelben egy munkalapon a sor és oszlopszám rögzített, a sor beszúrása nem jelent valódi új sort. Google Sheets-ben dinamikusan növelhetjük a sorok számát – egy ideig. A statikusan lefoglalt maximális méret, illetve dinamikusan változó méret előnyei és hátrányai ezeken a példákon jól szemléltethető. Képlethivatkozások másolásával a tömbök alul- és túlindexelésére, az ebből adódó problémákra is szemléletes példákat adhatunk.

Az adatbáziskezelők a tanult alkalmazások közül azzal tűnnek ki, hogy az adatokat felhasználás közben is a háttértáron tárolják, az adatelérés címszámítással történik. Ezzel szemben a tanult alkalmazásokra az jellemző, hogy a dokumentumokat egy folyamként kezelik, ahogy az I/O eszközökről érkező, oda küldött adatokat is. Az IoT és az egyre jellemzőbb online közös

munka szükségessé teszi a fájl fogalmának értelmezését, foglalkozni kell az adatok integritásának problémáival, a párhuzamos hozzáférés kezelésével. Az egyes alkalmazásokban gyakorlati tapasztalatot kell szerezni a többszörös hozzáférés szabályairól.

Példák fájlkezelés, adatkezelés szabályozására

- Jegyzettömb a dokumentumot a memóriába betölti, megnyitáskor minden megnyitott példány írható, mentéskor az aktuális példány felülírja a létezőt.
- Asztali Excel az xlsx fájl első példányát írhatóan nyitja meg, a további megnyitásoknál figyelmeztet és csak olvasható lesz a fájl. Ennek módosítása után más néven lehet menteni. Azonos gépen azonos nevű fájlokat nem lehet egyszerre megnyitni.
- Access adatbázisokat első lépésként el kell menteni, csak ezután lehet táblát definiálni. Ha a tábla tervező nézetben van megnyitva, akkor nem lehet benne adatot módosítani, adatot beírni. Lekérdezésekkel (a nem szerkesztett táblában) egyszerre többen dolgozhatnak, a zárolás rekord szintű.
- Online Excel fájlokat egyszerre többen is írhatják, látszik, hogy melyek az éppen szerkesztett cellák. A cellákban az érvényes adat az utoljára befejezett szerkesztés eredménye (mint a Jegyzettömb esetén).
- Érintőpad és csatlakoztatott egér együttes használata esetén milyen az egérmutató mozgása.
- Képernyőre kiírás (kimenet és log egy helyre vagy többszálú program esetén) lehet védett művelet – a megkezdett kiírás befejezéséig a következő kiírás várakozik – vagy ömlesztett, a két kiírás karakterei keverednek a képernyőn.

Hivatkozás, indirekció

A hardvernél tanult adatbevitel „gépbe” történik, a programozás során egy szoftver fogadja az adatokat azon belül függvények kaphatják meg, változók adhatják át egymásnak. A programozás tanulása során egyre hangsúlyosabb szerepe van az adat megszerzési módjának is: az adat átkerül (mozog), átmásolás útján vagy az eredeti helyére hivatkozással jut az adott eszköz (gép, program, függvény, változó) birtokába. A programozás tanításakor nem csak a programkészítést, alkalmazás készítését tanítjuk, hanem informatikát is, adatabsztrakciókat, adatmodelleket.

Példák a programozáson belül az informatikai tudás fejlesztésére:

- A multimédiás dokumentumokba a médiák beágyazása, illetve csatolása (hivatkozás).
- A táblázatkezelőben az adatok másolatát másolással vagy hivatkozással hozhatjuk létre.

- A függvény paramétereibe meghíváskor argumentumként az eredeti adat másolata vagy az adat címe kerül.
- A képernyőn megjelenő fájlhivatkozáson vagy ikonon keresztül közvetlenül érhető el a fájl, vagy a hivatkozás csak egy kapcsolat a fájl felé.
- Egy .lnk fájlra parancsikon készítésekor létrejövő .lnk fájl az elsőnek másolata lesz, vagy rá mutat? (A pointer másolata vagy a pointerre mutató pointer?) Lehet-e .lnk fájlokkal körbe hivatkozni?

Mindegyik esetben vizsgálandó, hogy a kapott adat módosítása az eredeti adatra milyen hatással van, megszűnhet-e (törlődhet-e) az átadás során az eredeti adat, az eredeti adat módosulása a használat közben kihat-e a kapott adatra, illetve hogyan befolyásolja az adat használata az eredeti adat használhatóságát, létét. Azaz, jelen példákban, a programozás tanítása során meg kell vizsgálni a másolással, az áthelyezéssel és indirekcióval kapcsolatosan a hitelesség, az adatbiztonság, a hatékonyság érvényesülését.

A hivatkozások minden formájánál kiemelt szerepe van az abszolút és relatív fogalmak értelmezésének. A fájl elérési útvonalának abszolút és relatív megadása évtizedekkel ezelőtt csak DOS-os fájlkezelési feladat volt, de napjainkra a multimédiás és online alkalmazások használatának természetes része a médiaelem globális helyének, illetve a hivatkozó objektumhoz viszonyított helyének a megadása. Táblázatkezelés tanulása során új értelmet (és felhasználási környezetet) kap ez a két fogalom, amely nem csak az adott objektum másolásakor vagy áthelyezésekor lesz releváns, hanem szabályok, számítások örökítésekor is, a „csináld ugyanígy” utasítás pontosításaként. Az adatbáziskezelés és programozás során az adatstruktúrákat alkotó adatok kapcsolata, az adatsorozatok, adathalmazok képzése a két fogalomnak újabb absztrakcióját igényli részben a szelektor, részben az indexelés, részben az összetett struktúrák bejárása kapcsán.

Programozáson belül haladó szintűnek számít, de adatbáziskezelésből alapvető ismeret a táblák kapcsolása, az idegenkulcsok használata. Ezzel egy összetett adatra (struktúrára) hivatkozunk egy egyszerű adat bejegyzésével. Ez az egyszerű adat programozásban lehet egy tömb indexe vagy egy objektum címe, esetleg egy pointer. Egyszerű adatok (hivatkozások) használatával relációkat adunk meg, ezek mentén indirekciók sorozata vezet például annak a kérdésnek a megválaszolásához, hogy „milyen a barátom autója üléshuzatának a színe”.

A relációkhoz hasonló „ereje” van a tömbökben az indexnek. Egyfelöl, a logaritmikus keresésnél is gyorsabb, ha a keresett adatnak tudom az indexét, mert címszámítással azonnal elérhető. Egy összetett adat egyik tulajdonsága alapján egy másik tulajdonság megadása (pl. „ki volt a leggyorsabb” kérdés esetén a sebességhez a név kikeresése) leghatékonyabban akkor adható meg, ha a tulajdonság (sebesség értéke) helyett annak indexét ismerjük. Hasonló megfontolásból, matematikában a függvények zérushelyeit, szélsőértékhelyeit és inflexiós pontok helyeit adjuk meg.

Algoritmus, értelmezés, számítás

A hagyományos programozásoktatás az algoritmusokra fókuszált, ezzel együtt a módszeres, procedurális, imperatív programozás jelentette a tananyagot. Az adatok végletekig (amennyire lehet integer típusra) történő absztrakciója, a matematikai problémák túlsúlya miatt az algoritmusok és programozás sokak számára száraz, nehéz tudomány volt. Az OOP, a robotika fejlődése, a grafikus megjelenítés, a webprogramozás és az IoT előretörése érdekesebbé tette a programozást, egyre könnyebb működő programot készíteni, de ez nem igényli algoritmusok kigondolását, a hagyományos értelemben vett módszeres programozást. Egy webes játék elkészítésének didaktikai elemzése [60] megmutatja, milyen „kerülő” utat jelent az algoritmusok oktatásában, ha először el kell készíteni az objektumokat. Hasonló eredményre vezet a padlórobotok programozása és – tapasztalatom alapján – minden OOP alapú játék- vagy programfejlesztésen keresztül a programozásoktatás. Az OOP lényege, hogy a programban szereplő objektumok önmagukon belülről vezérelve változtatják állapotukat. Így például a sorozatszámítás algoritmusának használata szinte hiba: nem „megyünk végig” az objektumokon, hogy kiszámítsunk mindegyikre egy újabb értéket, hanem minden objektum magában hordozza a kiszámítási módot, amit szükség esetén elvégez. Ez a gondolat látszik a VI/2 melléklet Eratosztenészi-szita példájának OOP megoldásán. Nagyon gyakori, hogy az OOP alapokon írt játékprogramokban a „szereplők” száma korlátos. Lényegében a játéktér kezelésével és egy-két szereplővel, avagy egy padlórobottal rengeteg érdekes játékot lehet készíteni úgy, hogy csak egy-egy vezérlési struktúrát használunk egy-egy függvény megírásakor. A játékhoz a mozgást állapotok átmeneteként adhatjuk meg egy óra segítségével, amelynek minden tick eseményére létrejön az előző-állapotból számított új állapot. Az állapotátmeneteket az egyes tulajdonságok újraszámításával lehet megadni, így a „főprogram” vagy egyszerű értékadások, vagy egyszerű feltételek teljesülésétől függő értékadások szekvenciája.

A fentiek némiképp korlátozott funkcionalitással egy animációkészítő (prezentációkészítő) alkalmazással megvalósíthatók. A blokkalapú programozási nyelv lehetőséget ad az állapotátmenetek finomhangolására, közvetlenebb vezérlésére, ami újdonságként jelenik meg a 2020-as tantervben, mint a felsőtagozat egyik eredménycélja. Az előző tantervekhez képest nagy változás, hogy a programozásoktatást OOP alapokon kezdhetjük, ahol az objektum előregyártott elem. Algoritmusokat már alsótagozaton is tanítunk, alsóban a szekvencia tananyag, felsőtagozaton ehhez társul az elágazás és a ciklus. Hagyományosan az algoritmizálás oktatását a „teafőzés" vagy hasonló hétköznapi algoritmussal vezetik be, amiről [121] cikkemben mutattam be, hogy az algoritmusban használt „adat" pontatlan definíciója miatt nehezebb az adat és algoritmus fogalmának, illetve ezek kapcsolatának megértése. A tantervben szereplő padlórobotok és a blokknyelvű programozás, a fizikailag megépíthető (például LEGO) robotok ezt a problémát kiküszöbölik, mert létükkel az objektum (az adat) definíciója is egyértelmű lesz. A hétköznapi algoritmusok tanításához mintaként tekinthetjük egy létező (vagy elkészíthető) robot, illetve objektum tulajdonságait, metódusait.

Az algoritmizálás tanítását a középiskolában a NAT szerint „típusalgoritmusok", illetve „egyszerű algoritmusok" tanítása követi. Ezek az algoritmusok már elemként használják a vezérlési struktúrákat, lényegében az egyszerű (elemi) programozási tételek (algoritmusminták) tanítását jelentik, amelyek sorozatokkal kapcsolatos műveleteket végeznek. A hétköznapi algoritmusok még pontos adatdefiníció esetén is nehezen használhatók ezeknek az algoritmusoknak az oktatásához, mivel az ember a mindennapi gyakorlatban nem úgy oldja meg a problémát, ahogy a gépek. Például egy csoportból a legmagasabb ember kiválasztásához nem kezdjük el elölről, sorban haladva nézni a csoporttagokat. Inkább úgy szemléltetném a kiválasztást, hogy fentről leeresztek egy lapot és akinek először eléri a fejét, az a legmagasabb. Hasonlóan a szőkét sem sorban haladva keresnénk, hanem „szkenneléssel", ránézve a csoportra észrevennénk.

OOP-vel nem könnyű olyan érdekes feladatot adni, amelyekhez egyszerű algoritmusok használata is kell, mivel ezek jelentős része statisztikához kapcsolódik. Tovább nehezíti a helyzetet, hogy a tanterv alapján vizuális blokkokról mindeközben át kell térni a karakteres kódolásra. OOP program esetén a programozási tételek alkalmazása előtt rengeteg előkészület kell. Ezt egy vizuális fejlesztőfelületű blokk nyelv esetén kattintásokkal meg lehet oldani, de a keletkező kód az osztálydefiníciók és objektumok létrehozása miatt hosszú; sok, egymásra is hivatkozó részből áll, aminek áttekintése nagyon nehéz. Ezen a ponton a programozási tételek alkalmazása előtt az előregyártott osztályok használata helyett meg kellene tanulni az objektumorientált szoftvertervezés alapjait, ami nem része a közismereti tananyagnak. Ráadásként

az iskolaváltás miatt egy csoporton belül nagy valószínűséggel különböző előismeretekkel rendelkező diákoknak kellene a tanult blokknyelvből karakteres kódolásra átállni.

A gyerekek számára fejlesztett blokknyelven történő programkészítés kódolás irányában történő továbbvitele valószínűleg nem vezet messzire, mert a nyelv a látványra optimális és nem a hatékonyságra. Nagyobb számításigényű programok, például nagy prímszám keresése, rendkívül lassan futnak. Természetesen a blokknyelvek fejlődésével elérhető, hogy a karakteres kódolással azonos hatékonyságú programot készítsünk ezen a felületen is, de ezzel együtt a használható blokkok száma is jelentősen nő, ami miatt a megfelelő blokk kiválasztása a listából nehezebb lesz. Keresés, szűrés hozzáadása viszont épp egy kódrészlet begépelését jelenti, ezért ergonómiai megfontolások miatt elkerülhetetlennek látszik a kódolás.

Egy OOP alapú általános célú nyelv, és ehhez egy grafikus felület kezelését is biztosító fejlesztő környezet megfelelő átmenet lehet a blokknyelv és kódolást támogató nyelvek között, mert ebbe át lehet emelni a blokknyelven tanultak OOP-vel kapcsolatos részét, ugyanakkor a kódolás már karakteres, megtanulható az egyes blokkoknak megfelelő kódnyelvi elem az adott környezetben. A grafikus fejlesztőkörnyezetben használt nyelv azonos lehet azzal, amit a konzolalkalmazásokban használunk. Az átmenetet segíti, ha a blokknyelvnek van kódnézete és az hasonló – a vezérlési struktúrák szintjén azonos – szintaktikájú, mint az új nyelv. Ebben a környezetben néhányórás átvezetéssel át lehet térni a konzol alkalmazások készítésére.

A középiskolai algoritmizálás és programozás témakör az imperatív, procedurális programozást jelenti. Fókuszában az algoritmus, a programozási tételek találhatók. A blokkprogramozás és OOP ismeretekre építés mellett fontos a többi témakörben is e témát előkészíteni. Az OOP programok továbbfejlesztése helyett a gyakorlatban, alkalmazásokban található megoldások megfigyelése, „utánzása” több és célravezetőbb lehetőséget ad a megértésre, begyakorlásra. Bár a kódolás tanítása a középiskolában, jellemzően a táblázatkezelés és adatbáziskezelés ismeretek után célszerű, az algoritmusok tanítását jóval korábban el kell kezdeni, ami jellemzően egybeesik a többi témakörben tárgyalt tartalmak informatikai vonatkozásainak oktatásával. A legfontosabb alapok: csere, keresés, összegzés, szélsőérték-kiválasztás, feltételes összegzés, rendezés, szűrés (kiválogatás).

Csere

Mivel a kezünk viszonylag összetetten működő szerszám, a benne lévő két eszköz cseréjét helyben végezzük. Ezt látja a gyerek és ezt utánozza. Nagyobb tárgyak cseréje esetén külön rá kell szólni, hogy előbb az egyiket tegye le… A csere algoritmusát több, tantervben szereplő alkalmazásban taníthatjuk: felcserélve elnevezett fájlok átnevezése; rasztergrafikában képrészletek

cseréje; szövegben formátumok vagy szövegrészletek cseréje, táblázatban cellák, sorok, oszlopok felcserélése. Emellett oktatóprogramok, számítógépes és kártyajátékok is segítik az algoritmus elsajátítását. A programozást előkészítő robotika – több robot esetén – szintén alkalmas az algoritmus gyakorlására, de a vektorgrafikus képszerkesztés, az animációs programok és a blokknyelven írt játékokban nem jelent problémát, ha két objektum „egy helyen van", azaz takarja egymást. Ezeknél az algoritmus három lépésének elvárása kényszermegoldás lehet.

Keresés, eldöntés, kiválasztás

A keresés algoritmusának tanítása a legnehezebb. Az első nehézséget az jelenti, hogy a keresés többjelentésű kifejezés. Ráadásul a különböző megoldásokat tanulni, gyakorolni kell. Például: keresem a zoknim, keresem az egyenlet megoldását, keresek egy vevőt az autómra, keresek egy idézetet. A keresés algoritmusához először az szükséges, hogy iterálható legyen az a tartomány, amiben keresünk. A keresések közé tartoznak a különböző struktúrák bejárási algoritmusai, például fabejárás, gráfbejárás algoritmusai. A szeriális adatok, sorozatok esetén is négyféle keresési algoritmust alkalmazhatunk: a címszámításos (keressük az 5. elemet), a lineáris keresést, a logaritmikus keresést és – nem gépi algoritmusként – a szkennelést. Más az algoritmus, ha szélsőértéket keresünk, illetve, ha egy adott tulajdonságú elemre van szükségünk. A keresés algoritmusának tanítása során a lehetőségekhez mérten sokféle keresési fajtára kell példát mutatni, meg kell vizsgálni, hogy a rendelkezésre álló információk és a kérdés függvényében melyik keresési eljárás lehet célravezető. Például a diákoknak értenie kell, hogy mely esetekben nem megoldás a Google™ keresőjének a használata.

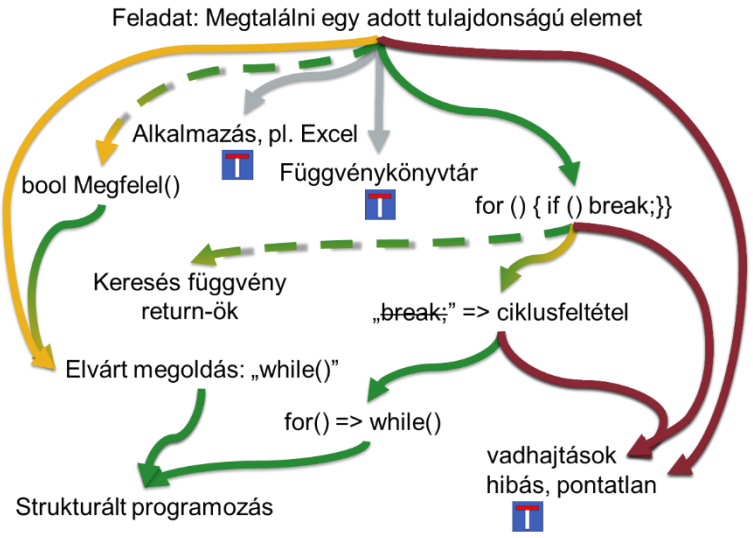
A keresési algoritmusokat érdemes különböző kereső eszközök használatának tanulmányozása során modellezni. Például modellezzük, hogy hogyan keresi meg a fájlkezelő a dolgozat*.* fájlt, hogyan találjuk ki a szükséges cipő méretét, hogyan keressük ki a névjegyek közül egy osztálytárs adatait, hogyan keressünk egy nyomtatott könyvben adatot és ugyanezt hogyan végzi egy számítógép egy elektronikus dokumentumban.

Többféle keresési mód tanulmányozása után célszerű a táblázatkezelésben használt kereső függvények működését modellezni. Ekkor kell tanítani a tartományra vonatkozó kritériumot (egy oszlop, egysor) és a két-, illetve háromféle algoritmust: a lineáris keresést és a növekvő vagy csökkenő rendezettségű adatsoron a logaritmikus keresést. Az Excelben magyarul HOL.VAN()-nak nevezett függvény működésének (különböző paraméterezések mellett a kapott eredménynek) az értelmezése kulcsfontosságú mind az informatika-, mind a programozástanulás szempontjából:

- A függvény használata a probléma deklaratív nyelvű megoldása.

- A keresési tartomány rendezettségétől függ a használható algoritmus.
- A paraméterezéstől jelentős mértékben függ az eredmény.
- A megoldás ellenőrzése, a tesztelés fontossága a tanulás során megtapasztalható. Több példa hozható a hibajelzés értelmezésére, a jónak látszó hibás megoldásra és az aktuális adatokkal jó, de elvi hibás megoldásra.
- Hatékonysági kérdések megfogalmazására ad lehetőséget. Az alapértelmezett beállítás a logaritmikus (tartományban) keresés. Miért? Miért nincs a keresés előtt beépítve a rendezés, hogy mindig logaritmikusan lehessen keresni?

A lineáris keresés algoritmusának tanítása nagymértékben függ a korábban tanult ismeretektől. A lehetséges tanulási utakról több cikkben, előadásban írtam, mivel a tanítási gyakorlatban jellemző probléma a különböző szempontok ütközése. A VI/1 mellékletben írtam arról, hogy a programozók között is tapasztalható a témával kapcsolatosan vita, ami az oktatásban komoly zavart okoz. A lehetséges megoldásokról a [57, 58] cikkekben írtam és több helyen előadtam (37. ábra).



37. ábra: A Lineáris Keresés Buktatói – diakép a megoldási utakról

Az elemzés hasznosnak bizonyult konkrét oktatási gyakorlatban is, mivel a tanulók megértették a saját megoldásuk, a társak megoldása és az elvárt megoldás közötti kapcsolatot, a megoldások mellett és ellen szóló érveket. Mivel ez az elemzés már túlmutat a közoktatás keretén, a megoldási módok közül – egyénre szabottan – a korábbi tanulmányok alapján célszerű választani: azt a megoldást tanítani, amihez a legközelebb áll, ami a tanuló számára „természetesen" jön. Azoknak, akik

1. robotikát tanultak, a végtelen ciklusból break-kel kiugrás lesz a természetes

2. blokknyelven programoztak, azok az OOP-re jellemző sok rövid függvény, eljárás írásához szoktak, nekik a keresés is lehet egy külön függvény, ami több helyen térhet vissza, náluk a több return adja a megoldást.
3. a deklaratív nyelvvel ismerkedtek és mögé képzelték az algoritmust, természetes lehet, hogy a keresési feltétel nem teljesülésig futó ciklust, a módszeres programozás megoldást választják.

Az 1. megoldást választók nagyobb, összetettebb feladatok esetén könnyen belezavarodnak a program strukturálásába. A 2. megoldást választók számára, kódolásra áttérve a sok kis programrészlet megtalálása, helyes összeépítése okoz nehézséget. Nekik át kell látni az osztályok rendszerét, számolniuk kell a függvények láthatóságával is. A 3. megoldásban viszont a logikában kell járatosnak lenni. Egy összetettebb keresési feltétel a helyes megoldás megalkotásának kritikus pontja.

Megjegyzés a 2. ponthoz: Ha a blokknyelv olyan, mint a Scratch, hogy a feltételes ciklusban a ciklus elején a ciklusból kilépés feltételét kell megadni, viszont a szöveg-alapú nyelven a hagyományos, ciklusban maradás feltételét váró while() vagy for() ciklus használható, akkor a keresés algoritmusának – sőt, a ciklus vezérlési struktúrának – tanulása is komoly kihívás. Már találkoztam olyan hallgatóval, aki a kilépés feltételét adta meg a for() ciklusban, mert úgy értette, hogy az kell. A Scratch és hozzá ebben hasonló blokknyelvek elterjedésével az átálláskor sokkal több figyelmet, energiát kell fordítani a de Morgan-azonosságok vizsgálatára, tudatos alkalmazására.

A közoktatásban elegendő az egyik megoldás gyakorlati ismerete, hiszen alkalmazni csak egyszerű esetekre kell tudni. Azonban a tanárnak fontos a többi megoldási módról is tájékozottnak lenni, mert ez biztosítja, hogy a többféle megoldási módot alkalmazók együtt tudjanak dolgozni, el tudják ismerni a másik megoldásban jártas társak erősségeit.

Szélsőérték-keresés, kiválasztás eredménye

A szélsőérték-keresés feladatokat táblázatkezelő alkalmazásokkal megoldva két tudásszintre bonthatjuk. Az érték keresése felsőtagozatban, az egyszerű statisztikai függvények használatával történik (MIN(), MAX()), a szélsőértékhez kapcsolódó további információk, azaz a szélsőérték helye, a szélsőértéket birtokló objektum megnevezése, más jellemzőjének meghatározása középiskolás tudás. Nem tudom, létezik-e kutatás, statisztika arról, hogy a számítógépes problémamegoldások körében melyik kérdés gyakoribb: mennyi a szélsőérték, hol van egy szélsőérték, melyek a szélsőértékhelyek, de mind a tanítási, mind feladatkészítési gyakorlatomban és

a mindennapok során is az a tapasztalatom, hogy egy szélsőértékhely meghatározása a gyakoribb, a szélsőérték értékének megadása másodlagos.

Táblázatkezelő alkalmazásban a MIN() és a MAX() függvény használatát követően gyorsan felmerül az igény a keresőfüggvények megismerésére. Például a mennyi a világcsúcs kérdés indukálja a ki a legjobb, ki tartja a világcsúcsot kérdéseket, amelyre a tanterv alapján csak évekkel később tud válaszolni a diák. Ekkor a keresőfüggvények használatával kapunk megoldást, amit sokkal nehezebb megérteni és helyesen használni. Lényegében az INDEX(HOL.VAN(MAX())) függvénykombinációt kell megérteni és alkalmazni tudni. A szélsőértékek mindegyikének megadásához még speciálisabb tudás szükséges: a helyes megoldáshoz speciális szűrőt kell alkalmazni, amelyben a szűrési feltételben függvényérték szerepel, a megoldás azonban eljárás (kigyűjtés) eredménye.

Adatbáziskezelő alkalmazásokban a mezőre számított MAX() és MIN() a táblázatkezelőkben megismert logikával számítható, de az értéket felvevő rekord többi adata kétféle úton adható meg, amely megoldásokat eltérő algoritmusokkal modellezhetünk. A táblázatkezelésben megismert algoritmust követve, egy segéd- vagy allekérdezésben meghatározzuk a szélsőértéket, majd ezt felhasználva megadjuk azokat a rekordokat, amelyeknek az adott mezője ezzel egyenlő. Másik gondolatmenet – és egyúttal másik algoritmus – szerint, az adatok rendezésével és első elem kiválasztásával, egy lekérdezésben megadhatjuk az eredményt. Bármelyik esetet választva, az SQL nyelv tulajdonságaiból következően az eredmény egy lista lesz, ami egyetlen előfordulás esetén egy elemet fog tartalmazni.

A deklaratív programozási nyelvek hátterében mindig egy imperatív végrehajtó motor működik. Vizsgálandó, hogy a „TOP” illetve a „LIMIT” megadása hogyan befolyásolja az eredmény megadását. Érdemes itt elgondolkodni azon, hogy szükséges-e az adathalmaz teljes rendezése, illetve – a rendezés algoritmusának tanításakor – érdemes arra kitérni, hogy az egyes rendezések közül melyek azok, amelyek az első néhány adatra is gyorsan elvégezhetők.

Az algoritmusok tanítása során a táblázatkezelésben használt megoldás algoritmusát vizsgálva látni kell, hogy legjobb esetben is a megoldásban két ciklus szükséges, egyszer végig kell nézni az adatokat a legnagyobb érték meghatározásához, egyszer pedig meg kell keresni az eredmény helyét. Ezután a válasz – az INDEX() függvény algoritmusa – már könnyen megadható tömbös tárolás esetén címszámítással, azonban dinamikusan csatolt sor esetén egy harmadik ciklus is szükséges lehet. Érdemes megvizsgálni a feladatok jellemzőit. A szélsőérték kiválasztása egyszerű adatsor esetén gyakran matematikafeladatot jelent, függvényvizsgálathoz kapcsolódóan szükséges. Itt a szélsőértékhely jellemzi a függvényt. Összetett adat, objektum

esetén a szélsőérték meghatározásához meg kell adni a szempontot, a tulajdonságot, azaz az adatra, az objektumra meg kell határozni a relációt. (Elméleti alapokon ez nem középiskolás tananyag, de a vizsgálat, a feladat elemzése igen.) Ezután a szélsőérték meghatározásában sokkal hatékonyabb az adat indexének vagy az objektum címének (referenciájának) megadása. Ebből a szempontból a szélsőérték keresése sokkal inkább keresés, mint összegzés.

Látható, hogy a szélsőérték-keresés az egyes problémamegoldásra használt környezetekben eltérő hatékonyságú algoritmusokkal történik. Az eredménycélként megadott programozási ismeret – az algoritmus megvalósítása formális programozási nyelven – azt jelenti, hogy az imperatív, procedurális programozás eszközeivel kell megvalósítani az algoritmusokat, ezért olyan programozási (fejlesztő) környezet kell, amely ezt támogatja. A konkrét eredménycél a szélsőérték helyének procedurális, függvénykompozíció nélküli meghatározása. A formális programozási nyelv tanítása nem (csak) a kód írásának elvárása, hanem a vezérlési struktúrák memória és futásiidő szempontjából hatékony alkalmazásának megtanulását célozza meg.

Feltételes összegzés, szűrés

Felső tagozatban elvárt az egyszerű statisztikai függvények használata: a megszámlálás, az összegzés, az átlag- és a szélsőértékfüggvényeket jelenti. Már ekkor, a felsőtagozatban tisztázni kell az összegzés és összeadás közötti különbséget. Mikor jó megoldás néhány cella összegét megadni, melyek azok az esetek, amikor a tartomány celláira (még, ha csak 1 vagy 2 cellára akkor is) végzett összegzés a helyes megoldás.

Informatikai tudást ad, későbbi problémák megoldásának a potenciálja, ha már a legelső gyakorlatok között, a függvények használatával való ismerkedés során tisztázzuk, hogy a függvények milyen adattípussal hogyan számolnak. A statisztikai függvények jellemzően számtípusokkal (ha dátumformátumú akkor is) számolnak, de létezik olyan változatuk is, amelyben a szöveget – 0 értékkel – illetve a logikai értékeket is figyelembe veszi. (A jelenleg használatos Excel verziók kapcsán a szoftverlokalizációs problémákra is érdemes kitérni: miért DARAB2 és ÁTLAGA; mikor MAX2 és mikor MAXA; miért nem tudják egységessé tenni.)

Az eredménycélokban szintén nem szerepel, de informatikai szempontból elvárható a hibakezelés ismerete. Például, a jegyek tantárgyi átlagánál az értékelés nélküli tantárgyakra kapott #ZÉRÓOSZTÓ hibajelzés helyett egy üres szöveg lehetővé teszi a további vizsgálatokat (a példánál maradva: azt, hogy a létező eredmények közül melyik a legjobb). Nem hibajelzés, de érvénytelen lehet az eredmény, ha olyan tömbön keresünk maximumot, amelyben nincs adat. Az ilyen és hasonló problémák kezeléséhez már felső tagozatban is elvárható a HA() függvény, esetleg célzott hibakezelő függvények ismerete.

A hibakezelés az, ami természetessé teszi, hogy egy cellában alternatívan két értéket jelenítsünk meg. Erre lehet építeni a feltételtől függő értékmegjelenítéssel és formázással kapcsolatos problématípusokat. Kezdetben a leggyakoribb alkalmazás a nem megfelelő adat esetén üres szöveg megjelenítése, ezt követheti a megfelelő cellák kiemelése formázással, a kiemelés kiterjesztése sorokra. A hibakezelés gondolatából következik, hogy adatsorokból a feltételnek megfelelő értékek külön oszlopban jelenjenek meg, míg a nem megfelelők sorában üres szöveg legyen – rejtetten ez a kigyűjtés algoritmusa –, így a feltételnek megfelelőkre lehet végezni összegzést. Ezzel a feltételes összegzésre lépésenkénti megoldást tudunk adni. A feladatmegoldásnak ez a formája a feladat szövegének elemzése, a probléma részekre bontása informatikai gondolkodás szempontjából fontos. Az ilyen típusú feladatok a feltételek és az utána alkalmazandó összegzések többféle lehetősége miatt egyértelműen kreativitást is igényelnek. (Ellenpéldaként, a ki_a_leg típusú feladat megoldását sokkal inkább be lehet magolni és utána el is felejtődik.) A feltételes összegzések elemi megoldásában az, hogy először kigyűjtöm a megfelelőket, nagyon helyigényes ráadásul egy idő után unalmas, ami felveti az igényt arra, hogy egy utasítással vagy függvénnyel lehessen megoldani ezeket a feladatokat. Az egy feltételt tartalmazó feltételes összegzésre táblázatkezelésben ötféle megoldás létezik (a magyar Excel elnevezéseket használva: a már bemutatott HA() függvénnyel „szűrés” majd összegzés, SZUMHA(), SZUMHATÖBB(), AB.SZUM(), {SZUM(HA())}). A feladatot adatbáziskezelőben megoldva a lekérdező rács és SQL nyelvű megoldás további két megoldási módot jelent.

A hétféle deklaratív megoldás csak a legegyszerűbb feladatoknál ekvivalens. Az adatmennyiség, a kérdések száma és a feltételek összetettsége alapján egyes megoldási lehetőségek hatékonyak, mások a megkötések miatt használhatatlanok. A megoldási módok taníthatóságáról, a tanítás sorrendjéről és feladattípusok szerinti hatékonyságáról a Guess the code of conditional summation [137] cikkben írtam. A gyakorlatban tapasztalt, különböző sorrendekben felépített tananyag helyett, érdemes szinte egyszerre tanítani mindegyik megoldási módot, mert a diákok egyéni ízlésétől, korábbi tapasztalatától függ, hogy melyik megoldási mód lesz számára a legjobb. Későbbiekben a feladatok nehezítésével, nagy mennyiségével a megoldási módok közül többet is kipróbálnak a diákok, de egyéni, hogy melyik megoldásokat preferálják. Az algoritmikus gondolkodást leginkább a tömbképlettel történő megoldás fejleszti [138], a többi módszerhez a tanítás során kiegészítésként célszerű kapcsolni a „géptől elvárt” algoritmust. Közoktatásban az elemi (egy feltételű) feladatok megoldása elégséges, a több – szűkítő, konjunktív – feltételű feladatok megoldása elvárható. A megoldási módoknak csak egy része alkalmas diszjunktív – „ilyen vagy olyan” jellegű – feltételek megadására, ehhez a megoldási módszerek szinte mindegyikében jártasság kell.

Az informatikaoktatás keretében, a függvények használatának gyakorlásával párhuzamosan kell a műveletvégzés algoritmusát is tárgyalni. Fontos ismeret, hogy az összegzés nem helyiértékenként történik, hanem kumulációval. Az összegzés algoritmusának hasonlósága a szélsőérték-keresés algoritmusával – eredmény kezdőértéke; minden adattal csinálok valamit, ami végeredményhez közelebb visz – később nagy segítség lesz a programozás tanulásánál. Egy-egy algoritmus blokknyelven meg is valósítható, például egy játék eredményeinek összegzésnél. A feltételes összegzések egyetlen függvénnyel történő megoldásában az előzetes kigyűjtés a kumulációval összeépíthető, azaz nem szükséges a kigyűjtött adatok tárolása. A feltételes összegzés algoritmusa hasonlít a szélsőérték-kiválasztás algoritmusához. Megfigyelhető, hogy a tömbfüggvénnyel történő megoldásban az összeépítés nem történik meg. Pont azért tömbfüggvény típusú a megoldás, mert a kigyűjtést külön elvégzi a memóriában létrehozott tömbbe. A beépített függvényeknél memóriaoptimalizálásra ad lehetőséget a ciklusok egyesítése. A megoldási módok elemzése hatékonysági szempontok felvetését teszi lehetővé, ami indokolja a megoldás formális nyelven történő – vezérlési struktúrákból építkező – megoldását.

A feltételes összegzésekkel párhuzamosan lehet tanítani a táblázatkezelésben a szűrést, valamint másolással a kigyűjtést. A SZUMHA() és SZUMHATÖBB() függvények összegzésmentes megfelelője helyben szűréssel megoldható. Az AB.SZUM() feladatok megfelelője az irányított szűrés. Mivel a táblázatkezelésben elsősorban függvényekkel dolgozunk (mert ezek automatikusan újra számítódnak), a szűrést a feltételes összegzés után érdemes tanítani, kiemelve azt, hogy a kigyűjtés függvénnyel mindig sok „nem látható” eredményt ad, ami memóriaigényes, míg a sorok kitakarása és másolás csak a megfelelő adatokat mutatja, kevesebb memóriát igényel, de nem frissül[60]. Adatbáziskezelésben a szűrés, a megfelelő rekordok kiválogatása az elsődleges, ezt „fejeli meg” az összegzés. Itt minden esetben eljáráshívásról van szó. A feltételes összegzés elvégzése adott mezők minden lehetséges értékére, a részösszegek számítása. A részösszegek számítására táblázatkezelőkben is van beépített eljárás, de sokszor függvények sorozatával oldjuk meg, ami sok adat és sok függvény esetén láthatóan memória- és időigényes újraszámítást eredményez.

Problémamegoldás informatikai eszközökkel témakörön belül, a táblázatkezelés és adatbázis-kezelés tárgyalásában be kell mutatni, hogy egy probléma többféle eszközzel is megoldható, de ennél is fontosabb annak sokféle szemléltetése, hogy a megfelelő eszköz kiválasztása nem

[60] Az MS365 Nagyvállalati Excelben létezik SZŰRŐ() és RENDEZ.ALAP.SZERINT() függvény. Emellett az alapértelmezett eredmény nem 1 adat, hanem egy tömb. Ezzel ez a táblázatkezelő még inkább tekinthető egy funkcionális programozási nyelvű fejlesztő környezetnek.

csak a probléma típusától, hanem annak méretétől – adatmennyiségétől, a megoldási mód számításigényétől –, az adathoz és a műveletvégzéshez való hozzáférés módjától, az eredmény aktualitásának fontosságától is függ. Az informatikai tudás része a hatékonyan használható eszköz kiválasztása, az eszköz hatékony (gyors, pontos) alkalmazása.

Példák feltételes összegzéshez eszközválasztásra

- Az adatok változását azonnal követő eredményekre van szükség, például költségtervezés – táblázatkezelő
  - SZUMHA() a legoptimálisabb akkor, ha egyetlen feltétel van és az az összegzendő adatokra vonatkozik. Pl. 100-nál kisebb értékek összege; töredékszavazatok összege. Hasonlóan ehhez, a többi statisztikai függvényre, pl. negatív értékek átlaga.
  - SZUMHATÖBB() a legoptimálisabb akkor, ha egy vagy több, de szűkítő (ÉS) feltétel mellett szükséges az összegzés és ezek a feltételek az egyes adatok értékére vonatkoznak.
  - AB.SZUM() a legoptimálisabb akkor, ha a feltételek egy része alternatív (VAGY) és a számítás egyedi, a feltételtábla elhelyezése nem okoz problémát, ha kell, a másolása könnyen megoldható. Pl. bukott vagy 3,5-nél alacsonyabb tanulmányi átlagot elérő diákok hiányzásának összege/átlaga.
  - {SZUM(HA())} tömbfüggvényes megoldás használata a legoptimálisabb akkor, ha a feltételekben az adatokra vonatkozóan számítás is szerepel. Pl. a hárommal osztható számok száma, összege, átlaga, maximuma.
  - Feltétel külön oszlopban kiszámítása, ebből egyszerű összegzés a legoptimálisabb, ha a feltétel összetett, más adatsoroktól is függ vagy az összegzés több típusa szükséges. Pl. előző naphoz képest melegebb napok száma, a melegedés átlaga, minimuma, maximuma.
- Nagy mennyiségű, esetleg több felhasználó által bővített, módosított adatok kezelése – adatbáziskezelő.
- Több táblázatban szereplő kapcsolódó adatokból statisztika készítése – adatbáziskezelő. Pl. a felvételin egy adott ponthatárt elért lányok első félévi tanulmányi átlaga.

A feltételes összegzés szövegalapú programozási nyelven történő megvalósítása a deklaratív megoldások mögé képzelt algoritmusok imperatív nyelven történő kipróbálását jelenti. Emiatt olyan nyelvet vagy a programozási nyelvnek olyan alrendszerét kell használni, amelyben a vezérlési struktúrákból építkezve, a megfogalmazott algoritmusnak megfelelően lehet megírni a

tanult függvények megfelelőit. Ahhoz, hogy a feladat érdekes legyen, a feltétel valamilyen gyakorlati problémával kapcsolatos legyen, a feladatokhoz tartozó adathalmaz összetett kell, hogy legyen. Táblázatkezelőben sorokba rendezett tulajdonságok táblázatával érdemes foglalkozni, de több esetben az oszlopok és sorok egyenrangúak. Adatbáziskezelőben a rekordok táblákban találhatók. Ennek megfelelően, a programozással megoldott feladatok adatstruktúrája egy- vagy kétdimenziós tömb vagy egydimenziós dinamikusan változó méretű tömb, amelyeknek az elemei adatstruktúrák vagy egyszerű adattagokkal bíró objektumok.

Amennyiben a táblázatkezelést – és ebben a függvények algoritmusainak elemzését – követi a programozás tanulása, akkor az egyes feladatok függvényként megoldása reprodukciós feladat lehet: „írjuk meg az adott feladathoz a SZUMHA() függvényt”. A megoldási lehetőségek párhuzamos oktatása és a megoldási lehetőségek algoritmussal majd programmal való modellezése nagyban segíti az informatika oktatási céljának megértését: a számunkra aktuálisan legmegfelelőbb eszköz kiválasztásához ismerni kell az eszköz működésének belső logikáját; az alternatív megoldások a konkrét feladat alapján rangsorolhatók, de nincs sem abszolút sorrend, sem univerzális módszer. Az eszköz kiválasztásának módja, az alternatív megoldások mérlegelése minta az alternatív alkalmazások vagy operációsrendszerek közötti választásra is. Az informatika humán területre való átvetítésével: társakkal való hatékony együttműködéshez nem elég a társak képességeit ismerni, meg kell érteni a belső motivációkat is.

Rendezés

Az adatok rendezése számos alkalmazás beépített eszköze. Fájlkezelőkben a fájlok és mappák tulajdonságai alapján, szövegszerkesztőben bekezdések szövege alapján lehet rendezni. Komolyabban táblázatkezelésben és adatbáziskezelésben tanítjuk, ahol a többkulcsos rendezés, illetve speciális rendezési beállítások miatt kap nagyobb hangsúlyt. Az algoritmizálás és programozás részeként nem jelenik meg tananyagként, azonban több szempont miatt nem szabad kihagyni.

- A kigyűjtött adatokat rendszerint valamilyen rendezett formában szeretnénk megkapni, ezért egy rendkívül gyakori feladattípus.
- A keresés módját jelentősen befolyásolja, hogy rendezett adathalmazban sokkal gyorsabban lehet keresni, kérdés, hogy nem rendezett adathalmaz esetén érdemes-e rendezéssel kezdeni a megoldást.
- Rendezési algoritmusból nagyon sokféle van, ezért akár a csoport minden tagjának önálló gondolata lehet. Az algoritmikus gondolkodást nagyon jól fejleszti a saját rendezési algoritmus kitalálása, elmagyarázása, illetve a társak magyarázatának megértése. (Esetleg: megoldások összehasonlítása, hol térnek el az egyes lépésekben)

A „saját” rendezési algoritmus nagy valószínűséggel egy már ismert algoritmus önálló megalkotása, amelynek helyességét konkrét programban célszerű bemutatni. A sikeres megvalósítás a programozástudás egyik mérföldköve. A „saját” algoritmus és a társak által készített vagy az ismert algoritmusokkal való összehasonlítása tudatosabbá teszi a „saját” algoritmus lépéseit. A saját gondolat összevetése mások gondolatával, az azonosságok és az eltérések elemzése nagyon fontos informatikai készség.

A rendezési algoritmusokhoz hasonló fejlesztő hatása lehet számos rendezéshez hasonló vagy azt használó feladatnak, mint például a hanoi-tornyai, kirakós kártyajátékok, tili-toli.

Logika

A vezérlési utasítások végrehajtása logikai kifejezések kiértékelésének függvényében történik. Logikát (jó esetben) már a bölcsődés korban tanulunk főleg implikációk formájában. (Ez felveti azt a kérdést, hogy a szülői következetesség hogyan hat ki a logikus gondolkodási készség fejlődésére, de ennek részleteivel nem foglalkoztam.)

1987-ben írt első szakdolgozatom [139] – az indirekt bizonyítás oktatásával kapcsolatban – jelentős részben szól a logikai készségek fejlesztéséről.

> *„Az elsőosztályos gyerekek … tételek kimondása és bizonyítása helyett szabályjátékokat játszanak. … A középiskola első osztályában találkoznak a tanulók először a logikával, itt tanulják meg, mit nevezünk logikai állításnak, mi az ítélet, a logikai függvény, megtanulják a konjunkció, a diszjunkció és az ekvivalencia fogalmát. … A középiskolás tananyagból azonban teljesen kimaradt az implikáció, azaz a következtetés mint logikai művelet, annak ellenére, hogy a gyakorlatban – feladatmegoldás, bizonyítás során – sokszor használják.”*

Nagyon régen kutattam ezt a témát, azonban a problémákkal napjainkban is találkozunk, az informatika térhódítása a témát hangsúlyosabbá teszi, oktatási vonatkozásainak alaposabb megismerését teszi szükségessé [140].

1. A köznyelvben használt logikai állítások nem felelnek meg a matematika logika kifejezési módjának. Például: „a belépés gyerek és nyugdíjas számára ingyenes”, „nincs semmi baj”.
2. A matematikában írt egyenletek jellemzően logikai kifejezések, a problémamegoldás, a bizonyítás során a matematikai logika használata jellemző. Informatikában meg kell különböztetni az értékadást (legyen egyenlő) a relációtól (egyenlő-e?); a végrehajtást az elemzésétől.

3. A ciklusfeltétel jellemzően a ciklusmagba belépés feltétele, miközben természetes gondolkodásunkban sokszor a kilépés feltételére gondolunk. Főleg a hátultesztelős ciklusnál feltűnő ez a gondolkodásmódbeli kétféleség.
4. Összetett logikai állítások tagadása szóban (fejben elvégezve) nagyon nehéz. Az előző két pontban szereplő esetekkel is összefüggésben, amikor már a kifejezendő állításban is pontatlanságok vannak, ennek a tagadása anyanyelvi eszközökkel még nehezebb. Például: kinek kell fizetni a belépésért, ha gyerek és nyugdíjas számára, valamint minden hónap első hétvégéjén ingyenes.
5. A logika formalizálásával a kifejezés pontosabbá válik, de könnyen elveszíti köznyelvi értelmét, ezért nehéz az eredmény összevetése a szándékkal. Míg a számítások eredményét nagyságrendileg lehet ellenőrizni, a logikai kifejezések eredményének nincs nagyságrendje, egy bonyolult kifejezés átalakításának eredménye teszteléssel lehetséges.
6. A Ha…, akkor… mondatszerkezet kétértelmű. Az egyik értelmezésben, az implikációban a hárompont helyén premissza és konklúzió szerepel, a másik értelmezésben, a vezérlési szerkezet feltétele (condition) és utasítása (statement) szerepel. Az informatika imperatív gondolkodásmódja a logika segítségével szabályoz, a három vezérlési struktúrából kettőben, az alternációban és a ciklusban a logikai kifejezés „vezérel”, ami más, mint a „megállapító” konklúzió. A [53] cikkben leírtak alapján e két fogalom zavarja egymás értelmezését. Emiatt, ahogy azt [57, 58] cikkekben feltártuk, a lineáris keresés strukturált algoritmusának valódi megértése nehéz.

Példa: Implikáció és vezérlés a lineáris keresésben

A: az i-edik elem létezik, azaz i< N; B: a sorozat i-edik tagja rendelkezik a keresett tulajdonsággal
ciklus amíg $A \wedge \neg B$

kilépett, mert $\neg A \vee B$

- $\neg A \vee B$ egyszerűbben egy implikáció: $A \Rightarrow B$. Jelentése: ha a sorozat létező elemét kaptuk akkor az rendelkezik a keresett tulajdonsággal.
- Mint vezérlésátadás, az „A”-t ellenőrzöm, mert az $\wedge$ rövidzár tulajdonságát így tudom érvényesíteni. Ha(A) akkor talált, különben nem talált.

A lineáris keresés deklaratív megfogalmazása az indirekt bizonyítás gondolatmenetét adja:

- Állítás: $\exists A \Rightarrow B$ (Van olyan elem, amelyik rendelkezik a tulajdonsággal).
- Bizonyítás: Tegyük fel, hogy $\forall A$ esetén $\neg B$ … (Jelentése: azt látjuk, hogy soha nem teljesül az adott tulajdonság.) A konklúzió: $\neg\exists A$ Ellentmondásra jutottunk. (Jelentése: a sorozatban nincs elem.)

7. A matematikai logikát sokféleképpen használja az informatika, de jellemzően módosított értelmezéssel. Például, az alternatíva a „vagy” művelet, ami az elágazás vezérlési struktúra jelzője lesz. A diszjunkció a „kizáró vagy” művelet, de a logikai kifejezések felírásában a konjunktív – „és” műveletekből építkező, szűkítő, metszetképző – forma ellenpárjaként adott diszjunktív forma a „vagy” műveletekből építkező, megoldási uniót képző kifejezési mód, ahol szó sincs az egyetlenséget jelző „kizáró” tulajdonságról. Nem hagyható ki a logikai műveletek értelmezése és használata során a rövidzár elve, amely a legalapvetőbb matematikai tulajdonságokat – kommutativitás, asszociativitás – írja felül és alapvető szerepe van a programok helyes működésében.

Az első két pontban megfogalmazott ismeret oktatása – napi gyakorlatként – a szövegértés és algoritmizálás gyakorlása keretében történik. A harmadik pont szükségessé teszi a logikai kifejezések formális felírásának tanítását a korábbi gyakorlathoz képest sokkal fiatalabb diákoknak. A negyedik és ötödik pontban már a formálisan megfogalmazott kifejezések használatában való megfelelő jártasság jelenti a probléma megoldását. A hatodik és hetedik pontban leírt problémákra a megoldás a közoktatásban a gondolkodási modellek szétválasztása. A bizonyítás, azaz a „belátás” nem vezérlés. A matematika és az informatika különböző modelleket használ problémamegoldásra, a definíciók a modellhez kapcsolódnak, a modellen belül érvényesek. Az informatika tudomány magasszintű művelésének része lehet ezen modellek összehasonlítása. A közoktatásban az azonosnak látszó fogalmak különbözőségére kell helyezni a hangsúlyt: programozásban az „és” nem kommutatív, a Ha…, akkor… vezérlési struktúra, amibe feltételt és utasítást adunk meg.

Programkészítés

A programozástudás egyik lehetséges értelmezése a programkészítés képességének megléte: akkor tudok programozni, ha képes vagyok programot készíteni. Az új tantervek, a fejlesztési tervek és a világtrend alapján a programozást egészen kis korban el lehet kezdeni tanulni, lényegében már az óvodában. Természetesen a korosztálynak megfelelő eszközökkel, ezért szerepel a tantervben a padlórobotok programozása. Ebben az értelemben a programozáshoz az kell, hogy

1. tudjunk utasításokat kiadni, amit a gép tárol, értelmez – programozásinyelv-ismeret;
2. az utasításokat át tudjuk adni a gépnek – fordítás (interpreter, compiler);
3. el tudjuk indítani a végrehajtást – futtatás;
4. ellenőrizzük, hogy a végrehajtás a szándékunknak megfelel-e, tudjuk módosítani az utasításokat – tesztelés, szemantikus ellenőrzés

5. hiba esetén értsük a hibajelzéseket (legalább a legalapvetőbbeket) – fejlesztőkörnyezet ismerete, szintaktikai hiba és eszközhiba felismerése;
6. ismerjük a hiba elhárításának módját – eszközismeret, eszközkezelés.

Padlórobotoktól a programozható építőjátékokon át a robotikáig

A legegyszerűbb programozható (oktató) eszközök a padlórobotok. Jellemző, hogy nincs külön fejlesztőfelület, hanem a roboton levő gombok egymásutáni megnyomása vagy egyéb fizikai eszközök egymásután helyezése jelenti a kódolást. Az első algoritmus, amit meg kell tanulni: bekapcsolás, szerkesztőmódra váltás. kódolás, kód lezárása, futtatás, szerkesztő mód… kikapcsolás. Ennek ismeretében lehet „alkotni" egyéb programokat, megfogalmazni algoritmikusan egy útvonal bejárását. A leggyakoribb hiba az elem/akkumulátor lemerülése (felkészült óvodás tisztában van azzal, hogy különböző játékaiba mennyi, milyen típusú elem kell, hogyan lehet tölteni az akkumulátort). A programozást nehezíti, hogy a „kódolást" nem szabad elrontani. Óvodásoknál még a tervezés is fejben történik. Egy átlagos program beírása gombok lenyomásával sokkal nehezebb, mint az iskolaérettséget vizsgáló sorminta rajzolása, mert nem ciklikus és ráadásul nem is látható, hogy egy adott állapot előtt mi lett kódolva.

Az „irányító panel" – a kódoláshoz használható kártyák, kockák és egyéb eszközök – használatával fizikailag kettéválik a fejlesztő és a futtató eszköz. Általános iskola alsó tagozaton már lehet tömöríteni a kódon, az ismétlődést számmal megadni. Ekkor már igényként jelentkezhet, hogy a programot – például azt, hogy 6 előre 1 jobbra – ne kártyákkal, hanem számítógépes alkalmazással írjuk és a megfelelő számú elem kirakása helyett elegendő legyen egy szám megadása.

Ezen a ponton a programkészítésbe algoritmikusan megjelenik a ciklus, de jelentősebb a fenti hat pontból a 2., az 5. és a 6. programozási részterületen a változás. Már nemcsak a robot merülhet le, hanem a mobil/tablet is. És a memóriája is megtelhet, lefagyhat, felbukkanhat egy nemvárt programablak… A fejlesztőeszközt is alaposan meg kell ismerni. A program áttöltése is okozhat problémákat: az adó adásra, a vevő vételre legyen állítva és stabil legyen a kapcsolat… A programkészítés folyamata összetettebb lett, a lehetséges hibák száma nőtt.

Az útvonalak programozása egy idő után nagyon unalmas, nem szólva az alakállandó robotokról. Az elemekből összeépíthető robotok programkészítés szempontjából nyelvi (1. programozási részterületen) bővítést jelentenek, a programban megjelennek az elemeket kezelő objektumok: motorok, szenzorok; finomabban paraméterezhetők az utasítások: a motor működésére vonatkozó egység, hossz és fok mértékek; a szenzorok adatai feltételek megadását teszik

lehetővé. És tovább bővül a hibalehetőségek listája: elemek kapcsolódásának hibái, sérült eszközök; figyelembe kell venni a mintavételezési és kvantálási tulajdonságokat.

A legismertebb programozható építőjáték a LEGO® Mindstorm, ami ütésálló, „kakaóbiztos", illesztési tulajdonságai nagyon jók, mérési és vezérlési pontossága tűrhető. Legnagyobb előnye azonban nem programozásban, hanem a passzív elemek sokféleségében rejlik. Legnagyobb hátránya pedig az ára. Robotok programozása szempontjából fontos, hogy az építőjátékokban az aktív (programozható, érzékelő vagy vezérelhető) eszközök jól védetten, burokban, modellelemben találhatók, valójában játékok. Az „igazi" aktív eszközök (a különböző mikrokontrollerek, motorok, LED-ek) a használati eszközökben is megjelenő szenzorok, kapcsolók. Ezekből robotok, vagy inkább eszközök készítése típustól függően, egyre komolyabb előismereteket igényel a matematika, a természettudomány, a műszaki és informatikai ismeretek (MTMI/STEM) minden részéből. Az errefelé történő továbblépés a micro:bit eszközkészlet használata, amelyben az aktív elemek modulárisak, könnyű az összekapcsolásuk, átépítésük, de kevésbé védettek és a passzív elemeket egyénileg kell előállítani.

Fejlesztőkörnyezet és nyelv

Amikor a padlórobotok irányítópultja helyett megjelenik a számítógépen futtatható fejlesztőkörnyezet, a hardveres problémák megsokszorozódnak. Informatikatanítás szempontjából ez elég nagy időveszteséget jelent, ezért érdemes a robotok, külső eszközök vezérlése helyett szoftveres problémákra áttérni. A felsőtagozaton megjelenő blokknyelvű programozás alkalmas robotok programozására, de épp ennyire megfelel a számítógép beépített vagy alapértelmezett inputjain érkező jelek vételére, az outputjain keresztül vezérlésére is. A fejlesztő és futtató környezet ugyanazon az eszközön van, így a fordítás (2. programozási részterület) kevesebb hibalehetőséget rejt, az 5. és 6. részterület a használt számítógép és használt fejlesztőkörnyezet hibajelzéseinek ismeretéről szól.

A blokknyelvek használatának divattá válásával az adott korosztályra tervezett és éveken át tanított Logo a hazai oktatásban háttérbe szorul. Míg Logoban a kódolás szöveges, a blokknyelveken a blokkok előregyártott vizuális kódelemek, amelyek kapcsolódása vizuálisan ellenőrizhető illeszkedéssel történik. Ezért a szintaktikai hibák valószínűsége jelentősen kisebb, csak a paraméterek típusára vonatkoznak, ami megkönnyíti a program készítését, ugyanakkor kevesebb jártasságot ad programkészítés problémáinak megoldásában. A Logo kiváltásával együtt járt, hogy algoritmusok tekintetében a rekurzióra (bár lehet, de) nem kell kitérni, cserébe a strukturált programozásra jobban felkészít a blokknyelv.

A középiskolában tanított szövegalapú programozás minden tekintetben nagy váltást jelent a blokknyelvű programozáshoz képest. A blokkokkal programozás és a szöveges programozás között majdnem akkora a különbség, mint a cukrászdában vásárolt tortalap és a lisztből, tojásból, egyéb hozzávalókból sajátkezűleg készített torta között. Mutató eszköz használata helyett gépelni kell. Egy elem beillesztése helyett minden karaktert be kell írni, jó sorrendben, az is számíthat, hogy kicsi vagy nagybetűvel írjuk. A blokknyelven jól programozó diákok számára az áttéréskor zavaró, hogy a program begépelése lassabban megy. A gépeléssel a hibák száma is jelentősen megnő. A szemantikai hibák kiszűrésében segít a fejlesztőkörnyezet, de meg kell tanulni „kommunikálni vele”. Az átállást jelentősen kellemesebbé (könnyebbé) teszi, ha a fejlesztőkörnyezet hatékonyan támogatja a kódkiegészítést, snippetekkel segíti a gyors kódolást.

A szöveges programozási nyelveknek sokkal több eleme, alapkifejezése, beépített eljárása van, jelentősen kevesebb az alapértelmezett érték, művelet. Ezért több kódrészletet kell megtanulni, amelyek egy része kezdetben érthetetlen. A mintakódot sokkal könnyebb másolni, ami átsegíthet a kötelező kódsorok gyors reprodukálásán, de károsan hat a tanulásra. A kész kódrészleteket is célszerű gyakran végigírni, mert másképp a részletek megtanulása elmarad.

Példa másolható, de begépelendő kódokra:

- C nyelvben stdio.h, mert nem stbio.h és nem is studio.h; math.h, ami nem mat.h; scanf(″%d″, &a)
- C# nyelven namespace{class Program{}}; static void Main() – más nyelveken másképp

A kód olvasása – megírt kódok értelmezése – általában nehezebb, mert a színezés kevésbé segít: a blokkban lényegében a megjelenített struktúra háttérmintázatának színezése jelenti a segítséget, a szöveges nyelvben viszont a kulcsszavakat, típusokat karakterszínezéssel emelik ki.

A futtatás többféle módját érdemes megismerni:

- fordítással együtt debug módban,
- töréspontoknál megfigyelve a változók állapotát,
- release módban a fejlesztő környezetből, valamint
- parancssoros fordítással és futtatással.

A programkészítés igazi élménye azonban a kész program futtatása. Pontosabban az, amikor a kész programot mások futtatják.

A szövegesen kódolt programkészítést nem csak a blokknyelvű programozással lehet előkészíteni. Informatikai gondolkodás szempontjából is érdemes elemezni, hogy miben különböznek vagy egyeznek meg a blokknyelvek, a fejlesztőkörnyezetnek, a programozási nyelvek, a játékkészítő, illetve a dokumentumkészítő alkalmazások.

A Klik & Play (1994-ben kiadott) játékkészítő alkalmazás „programozásmentes”, személyközpontú OOP alkalmazás, amiben azért nem kell programozni, mert az eljárásokat is készen kapjuk, csak ki kell választani. A Game Makerben a GML script nyelven lehet az objektumok eseményeihez kódot írni. A Game Maker játékfejlesztő szoftver, benne a GML egy nagyon kis programozási nyelv. A Unity (valós idejű 3D fejlesztőkörnyezet) ehhez hasonló, de C# nyelven lehet programozni. A MakeCode egy fejlesztőkörnyezet, amiben blokknyelven és JavaScriptben lehet programot írni mikrokontrollerekre, MineCrafthoz. A Scratch a leírások alapján egy programozási nyelv. Valójában fejlesztő környezet, ami a blokkokból JSON projektet készít, amit tömörített kódfájlba ment. A Visual Studio, az Eclipse, a Code::Blocks… fejlesztő környezetek, amelyekben egy vagy (inkább) több nyelven lehet programozni. Ezek a nyelvek jellemzően szövegalapú nyelvek, de létezik grafikus moduljuk is, amit egyes fejlesztőkörnyezetekhez integráltak. Például az állapotgépek működését leíró YAKINDU Statechart Tools az Eclipsbe integrálható, a Nassi-Shneiderman diagram a Code::Blocksban található. Ez utóbbi továbbfejleszthető lehetne blokknyelvre. Mindkét példára jellemző, hogy a grafikusan készített programot szövegalapú nyelvre át lehet alakítani, visszafelé viszont nem alakíthatók. Erről az oldalról nézve, az újabb Unity egy, a Visual Studioba integrált modul.

Visszatérve Klik & Play-re, ez a játékkészítő alkalmazás a bemutatókészítő alkalmazásoktól annyiban különbözik, hogy az objektumok egymásra is hatnak, például ütköznek. A különböző események állapot- vagy mozgásváltozást eredményeznek a PowerPointban is. A bemutató pptx fájlja tömörített formában, xml kódban tárolja a tartalmat, a formát, az animációt. Ebben az értelmezésben a PowerPoint egy fejlesztőeszköz és interpreter, a pptx állomány pedig a kód. Nem blokknyelven, de grafikus környezetben, WYSiWYG nézetben kódoljuk a bemutatónkat. Az animációtól eltekintve hasonlót mondhatunk a többi dokumentumkészítő alkalmazásról is, csak a nyelv más: nem programozási nyelv, hanem dokumentumleíró nyelv: XML, HTML, CSS, TEX vagy binárisan kódolt utasítás a megjelenítendő tartalomra és formára.

Nem informatikáról lenne szó, ha nem kellene azonnal pontosítanom: a HTML5 nem (csak) dokumentumleíró nyelv, webes dokumentumok mellett webes alkalmazások, programok készítésére is alkalmas. A HTML dokumentumleíró nyelvből programozási nyelvvé fejlődött.

Mivel a weblap az internet dokumentuma, sokféle hardveren, operációs rendszeren futó sokféle böngészőnek kell ugyanazt megjelenítenie, ezért jól látható a fejlesztőkörnyezet, az adott nyelven írt kód és a futtató (interpretáló) eszköz és szerepük is. A dokumentum kódja ezért a webes dokumentumok esetén „közismert". A HTML (és CSS) oktatásával a dokumentumleíró nyelvek tanulása közben a programkészítést is – akarva vagy akaratlanul – előkészítjük. A kódolás, a szintaktika ellenőrzése és javítása, a futtatás böngészőben (mint interpreterrel), a tesztelés és a szemantikai hibák javítása a programkészítéshez nagyon hasonló módon történik. A fejlesztő környezetben írjuk a kódot, amit a futtató környezet értelmez. A weblapkészítés esetén a fejlesztőkörnyezet lehet egy egyszerű szövegszerkesztő, egy WYSiWYG weblaptervező eszköz, egy portálba integrált tartalomkezelő – amely saját jelölőnyelvvel is rendelkezhet és lehet menüvezérelt is. Az előállított kód a fejlesztés eszközétől független, a különböző eszközök az emberi munka, az önkifejezés hatékonyságában, a szemantikai és szintaktikai hibák kezelésében adnak eltérő támogatást. Egy WYSiWYG weblapszerkesztőben demonstrálható, hogy a kijelölt szöveg formázásához megnyomott gomb hatása a kódban a megfelelő nyitó- és záró tag beszúrását jelenti. A kijelölés meghatározza a blokkot, nem lehet hibásan elrendezni, nem lehet elírni. Ugyanez történik az összes dokumentumkészítő alkalmazásban, de még a pixelgrafikus képszerkesztőkben is. Mivel általában csak formátum – passzív tulajdonságok, attribútumok – beállításáról van szó egy dokumentumban, ezért megoldható menüből. Amint aktivitása lesz a dokumentumnak, azt a fejlesztőkörnyezet külön szerkesztőjében adjuk meg, külön eszköz kell a hibakezelésre (ami nem WYSiWYG). Például a bemutatókban az animáció és áttűnések szerkesztése, a videószerkesztőkben a timeline, a táblázatkezelőben a képletszerkesztő és a hibakereséshez a képletkiértékelő eszközök. Mindegyiknél megfigyelhető, hogy a kódolást vizuális elemekkel segíti a fejlesztőprogram.

Figyelemre méltó, hogy WYSiWYG weblapszerkesztővel a hatékony munkához a kódnézet is szükséges. Táblázatkezelő programban függvényvarázsló, függvénytündér, azaz függvényszerkesztő panel segíti a szintaktikai hibától mentes függvény létrehozását, de a függvények szöveges beírása a kódkiegészítő támogatással sokkal hatékonyabb az összetett függvények létrehozásakor, szerkesztésekor. Adatbáziskezelés feladatokat táblázatkezelőben is meg lehet oldani, ilyenkor az adatbázis-kezelő alkalmazások lekérdező rácsához hasonló módon használjuk a táblázatkezelőt. Adatbáziskezelőkben lekérdező ráccsal könnyen „összekattintgathatók" az egyszerű lekérdezések, de a generált SQL kódban a megnevezések és a feltételek zárójelezése rendkívül redundáns. A programkészítésben is ugyanez várható. A vizuális fejlesztőeszközök redundanciával segítik a programkészítést. A blokknyelvek – úgy tűnik – nem programozási

nyelvek, hanem a fejlesztőeszköz kódmegjelenítő sablonjai, képi megjelenése a „snippet”-eknek. A blokknyelv szövegalapú kódolás lehetőséggel kiegészítése nem hozzáadott képesség, hanem kikerülő hozzáférési lehetőség ahhoz az implementációhoz, amelyet a blokkok alapján generál a fejlesztő környezet. Ebből következik, hogy egy adott bonyolultság esetén a köztes fordítási lépésekben nehezebb lesz a kód optimalizálása, korlátozott lesz a funkcionalitása.

Az informatikaoktatás során a számítógépes problémamegoldást kétoldalról közelítjük. Egyrészt a számítógépes alkalmazásokkal előállított termékek a számítógép funkcionalitását egyre jobban kihasználják: kezdetben statikusak majd animáltak végül némi intelligenciával is rendelkeznek. Másik irányból a megoldások minőségében az előregyártott intelligens komponensek összeillesztésétől haladunk a komponensek egyedi elkészítése irányába. A két szál találkozása a vezérlési struktúrákból építkező, egyedi, szöveges programírás, a fordítás, a futtatás, a tesztelés és hibajavítás képessége.

Programozás

A programozás jóval több, mint programkészítés. A programozásnak része az adat- vagy objektummodellezés, valamint az algoritmus tervezése is. A programkészítés a programozás fizikai megvalósítása, miközben a programozás jelentősebb részben szellemi tevékenység. Jellemző, hogy a *Programozási alapismeretek* (ELTE IK) tárgyat és a *Programozás alapjai* (BME VIK) tárgyat egymáshoz képest nagyon eltérő, a tantárgy tekintetében lényegében azonos elveken, de többféle nyelven tanítják – C++ és Pascal, illetve C és Python – a fejlesztőeszközre pedig több alternatívát is kínálnak.

A „világ” azt igényli, hogy mindenki tudjon programozni. A NAT eredménycéljai között viszont nem szerepel, hogy a diák tudjon programozni, csak az egyes résztevékenységeket kell tudnia, a programozáshoz szükséges fogalmakat kell értenie, az algoritmusokat kell ismernie. Ez kevés, LAU^{L}1-3 szint. Nem elég érteni és ismerni, a szükséges pillanatban fel kell ismerni, hogy épp melyik ismeretre van szükség és alkotó módon kell tudni használni. Csak az alapokat, de azt LAU^{L}5c szinten kell tudni.

Programkészítési minimum

A programozást a táncművészethez hasonlítva: a közoktatásban meg kellene tanulni járni. A tanulónak képesnek kell lennie egy adott feladat vagy probléma esetén meghatározni a kapott adatok típusát, a megoldáshoz szükséges adat- vagy objektum-struktúrát. Meg kell tudni fogalmaznia a vezérlési szerkezetek szintjén a megoldás algoritmusát és a kimenetnek, az eredménynek formáját. Ezek alapján el kell tudnia készíteni a programot (azt a programot, ami a feladatot

megoldja), tudnia kell a megoldását tesztelni, a hibákat önállóan kijavítani. Tudnia kell a programjához felhasználói dokumentációt készíteni, amelyben leírja a program működésének feltételeit és korlátait, a kezelés módját, továbbá tudnia kell publikálni a kész programot, illetve kezelni a kódoláshoz, a fejlesztéshez szükséges forrásokat.

A programozástudás „tipegő” szintjének jellemzői:

- a feladat megoldásához néhány elemi adat vagy string kezelése szükséges,
- az algoritmus bonyolultsága az egyszerű programozási tételeknek megfelelő
  - for(){if(){}} vagy
  - több ciklus egymás után, vagy
  - ciklusmentesen több feltétel kombinálása;
- a feladat önálló megfogalmazását (pontosítását) követően a megoldás önálló megtervezése;
- kódolás;
- a kódolási hibák önálló javítása, futtatással tesztelése;
- az eredmény értelmes megjelenítése;
- az elkészült program bemutatása, a program publikálása felhasználási tájékoztatással;
- beszámoló az adatmodell az algoritmus és kód bemutatásával a megoldás során felmerülő problémákról és döntésekről.

A programozás „tipegő” szintű teljesítése azt jelenti, hogy a tanuló programozással teljesen önállóan megoldott egy feladatot, ezzel tapasztalatot szerez arra vonatkozóan, hogy képes programozni, programozási problémát megoldani.

6.2
6.2.4
6.3

X. Informatikaoktatás módszerei, eszközei

A *Programozási alapismeretek* (ELTE IK) és a *Programozás alapjai* (BME VIK) tárgyak tananyagfelépítésének és oktatási módszerek vizsgálata során tapasztaltak a disszertáció fontos adaléka, de a részletes elemzés túlmutat a disszertáció keretein.

X/1. Sok kicsi LAU sokra megy

Előrehivatkozás

A leglátványosabb probléma a C-nyelv specifikációjából adódik: a beolvasáshoz a változóra a címével kell hivatkozni, scanf(″%d″, &a). A beolvasást az első órák egyikén kell tanítani, míg az indirekció csak jóval később, a 6. héten (közoktatási tanórákban mérve évekkel később) kerül sorra. Az előre-hivatkozás „*& kell a változó neve elé, ennek okáról majd részletesen lesz szó*” elkerülhetetlen. Úgy tapasztaltam, hogy ez az egyik oka annak, hogy a C-nyelvet a kezdők számára nehéz, nehezen tanítható programozási nyelvnek tartják. Egy jelölést úgy kell megtanulni, hogy a tanuló kezdetben nem érti, hogy mit jelent. Más nyelvek esetén a nyelv „emberibb”, kényelmesebb írni, azonban nem érzékelhető, hogy a beírt kód hatására a gép valójában mit csinál. Azaz a programozásnak – adatok szempontjából – nem az alapjait tanulná a hallgató. Tervezés szempontjából teljesen általánosítható a jelenség: a LAU^{L}5 megelőzi a LAU^{L}1-3-at. A tanítás során az ilyen típusú kényszereket nagyon alaposan kell tisztázni. Hangsúlyosan ki kell emelni, hogy az adott dolgot „most még nem kell érteni; ne akard érteni, csak használd; <ekkor> fogjuk tanulni”. Természetesen minden ilyen kereszthivatkozás gyengíti a logikus tanulást, ezért a tervezés során törekedni kell a „majd később” elemek minimalizálására.

Ritkán előkerülő, de fontos tananyagelemek

A „ritkán használt, de fontos” tudáselemek egyike az `enum`. A tananyag felépítésétől függően 2–4 hét (közoktatási tanórákban félévek) is eltelhet két használat között. Lehet úgy tervezni, hogy első előforduláskor megtanítjuk, majd erre hivatkozunk, de gyakorlatilag addigra elfelejtődik, taníthatjuk újra. Jellemző, hogy az első alkalommal – vagy bármikor – még nem is akarjuk mérni a tudáselem meglétét, nem szerepel a dolgozatban. Az `enum` megjelenhet a `const`, a `struct`, az `union` fogalmak mellett a `switch` párjaként az állapotgépek témánál. Van, ahol kiegészítés, van, ahol szokásos, de elhagyható a használata. Érdemes azonban minden esetben újra megtanítani, minden esetben más, újabb kapcsolódást véve a fogalomhoz. Ez azt jelenti, hogy LAU^{L}1 után újra LAU^{L}1 szinten ismerkedik a fogalommal a tanuló, egy-egy vonatkozása tekintetében ki is próbálja (LAU^{L}1–3), de amíg ezek az ismerkedések nem érik el a LAU^{L}5a

szintet, addig nincs mit számonkérni (még beugróban sem). Ezzel a módszerrel lehet előkészíteni nehéz, sok kapcsolattal bíró fogalmakat is.

Alternatív megoldások

Az alternatív megoldások újabb tervezési feladatot adnak. A for ciklus C-beli megvalósítása valójában nem számlálós, hanem összevont feltételes ciklus, így bármikor használható a while helyett. Azért vannak esetek, amikor a while ciklust használják a mérnökök is. Például a végtelenítésre, while(true) formában. Cikkek [pl. 143] szólnak arról, hogy melyiket jobb előbb tanítani (az elágazást is hozzávéve). Tanítás során a for ciklushoz kapcsolódik a vessző operátor, amit leginkább a for ciklusfejében használnak, de szokás az adatkérés és beolvasás között is. Az első eset ritkán fordul elő (strukturális szempontból kérdéses, hogy szükséges-e a ciklus bonyolítása ilyen módon), a második ízlés dolga, de az oktató rendszeres példamutatásával szokássá teheti. Az is lehet, hogy kihagyjuk a vessző operátor tanítását, de akkor teljesen ismeretlen maradhat a hallgatók előtt és hibaként értelmezik a használatát. Ugyanebbe a körbe tartozik a `break;` használatának szokása is. Minden, adott feltételre bekövetkező megszakítás helyettesíthető az adott feltétel tagadása esetére írt elvárt művelettel, tehát a `break;` használata kikerülhető.

További példák alternatív megoldási lehetőségekre:

- azonos típusú változók deklarálása egy utasításként, illetve külön;
- C++ nyelven a struct és class közötti választás, template-ekben a typename vs. class használata;
- rendezési algoritmusok közül választás (ne mindig szélsőérték kiválasztással, ne mindig buborékosan);
- megjegyzésekben a /* … */ és // használata;
- C#-ban a kiírásokban a helyőrző használata a paraméterek beillesztéséhez, illetve a szövegösszefűzése.

Rutinok

Az alternatívák kipróbálása után a tapasztalat alapján a hallgatók kialakítják saját szokásaikat, egyes megoldásokban rutint szereznek. Eközben az oktató folyamatosan a sokféleséget gyakorolja, azaz a rutin helyett az motiválja, hogy ne maradjon ki az a megoldás se, amit ő nem szokott választani. A legtipikusabb problémák az operációsrendszer, az alkalmazói szoftverek, a programozási környezetek és nyelvek, a billentyűzetkiosztás választása során jelentkeznek.

Mindenki tudja, hogy többféle operációsrendszert kell tudni használni, ezért többfélét kellene tanítani. Mondjuk Windowst és Ubuntut… De az operációsrendszerek kezelőfelületei között meglévő eltérések olyanok, hogy a párhuzamos vagy felváltva történő használat megakadályozza a rutinok bevetését, lelassítja a munkát. Ha minden művelethez el kell gondolkodni azon, hogy hogyan hívják azt az adott rendszerben, hogyan néz ki, hol található, miközben még a tanulóra, a konkrét tanítandó tananyagra is figyelni kell, akkor hamar feladja az oktató. Ezért jellemző, hogy az alternatív megoldásokból az oktató valójában csak az általa megszokott megoldást tanítja. Azonban ezeket a szelekciókat, döntéseket tervezni kell és az alternatív lehetőségekre, ahol lehetséges, utalni kell.

Másik oldalról, ha a tanuló alternatív megoldást alkalmaz, ahhoz tudni kell alkalmazkodni, már a tervezéskor át kell gondolni, hogy az alternatíváknak milyen következményei lehetnek, a tanuló megoldásánál mire kell odafigyelni. Az operációsrendszer választása esetén például arra kell figyelni, hogy a későbbiekben használt alkalmazások alternatívái is léteznek-e. A veszszőoperátor alkalmazása esetén pedig arra, hogy – bár érdekes lehet mindig azt használni – a program működését jelentősen bizonytalanná teszi.

Az informatikai tudásunk haszna, hogy rutinokat alkalmazunk [26], ezt oktatóként sem adjuk fel. Emellett az alternatív megoldásokat be kell mutatni és azokat a megoldási lehetőségeket is értékelni, elemezni kell, amelyeket nem használunk rutinszerűen. Sok esetben a használt megoldásunk nem a legoptimálisabb, de mivel rutinosan használjuk, hatékonyabb, gyorsabb lehet számunkra, más megoldásokhoz képest [27].

Egy jellemző példa erre a változók névválasztása: A hosszú változónevek leírása hosszú, az egybetűs változónevek semmitmondók. Az egyes nyelvekben a változónevek formáját szokások szabják meg, de egy-egy cég vagy projekt közössége saját szabványt szokott alkalmazni. Az egyéni vagy közös szokásrend néhány hetes alkalmazása rutint eredményez. Ezt a rutint akkor is könnyebb tartani (például a hosszú, leíró változónevekkel), ha az adott környezettől eltérő helyzetben van szükség a változónév megadására.

7.1.1

X/2. A hozott tudás

A BME VIK-en szerintem, az egyik legjobban felépített tárgy a *Programozás alapjai 1* (és *2*). Mégis, sokaknak (2016-2018 körülbelül 30% [147]) nem sikerül a teljesítése. A sikertelenségnek többféle oka is van, de az egyik biztosan a lemaradás a haladásban, illetve ezzel szorosan összefüggően a felkészületlenség. A kutatásom arra is irányult, hogy megtudjam, milyen képességek és tudáselemek szükségesek a tantárgy sikeres tanulásához.

A tananyagot elemezve látható, hogy a nulláról felépítettség a programozásnak csak egy részére, egyfajta értelmezésére vonatkozik, amely az adatot, az objektumot helyezi középpontba, a programtervezés nem tananyag az első évben. Az algoritmusok megfogalmazásának alapja: „ahogy csinálnád a gép helyében". A felvételhez nem elvárás a programozástudás és a vizsgált időszakban a mérnökinformatikára jelentkezőknek a matematikából (emelt szinten) és fizikából javasolnak alapos felkészülést. Az informatika – főleg az alkalmazói oldala – nem szükséges előismeret. Főleg fizikatanári tapasztalatom és a tapasztalt tanítási szokás alapján azt gondolom, hogy a fizikatudás azért fontos, mert a hagyományos fizikatananyagban a szöveges fizikafeladatok megoldása lényegében a programtervezéssel azonos módon történik: a leírás alapján kigyűjtjük az adatokat, képlettel modellezzük a fizikai összefüggéseket; „ezután már csak matek". A programozásban „ezután már csak kódolás" [32]. A programozás alapjai arról szól, hogy a kigyűjtött adatok hogyan lesznek a gép számára kezelhetők, a képletek helyett hogyan írunk algoritmusokat.

A fizikaoktatás fókusza a Dér-Radnai-Sós feladatgyűjteményről a kísérletek, jelenségek értelmezése felé fordul, képletek mellett szimulációs programok színesítik a tananyagot. A XXI. század diákja számára – például – az „elhanyagolható légellenállás" sokkal könnyebben értelmezhető, mint a XX. század diákja számára, mert számos játékprogramban el van hanyagolva a légellenállás, a súrlódás, sőt, néha még a gravitáció is. Napjainkban nem a program hasonlít a fizikai világhoz, hanem a fizikai világ egyes programokhoz. Korábban a fizika absztrakciója vezetett a programkészítés igényéhez, de ma már dominánsabb, hogy az informatika segíti a fizikai (és virtuális) világ megértését.

Az elemzés mellett kérdőívvel felmértük a hallgatók belépéskor meglévő ismereteit, a válaszoknak az első féléves programozásban eredményekkel vett korrelációját vizsgáltuk 2015-2017-ben a BME mérnökinformatikus hallgatói körében és 2016–2018-ban az ELTE *Programozási alapismeretek*et tanulók körében. 2017-ben a 2015-2016-os eredményeket mutattuk be a DidMatTech konferencián [148]. A kérdőívre adott válaszok és a félévi eredmény alapjául szolgáló pontszám alapján az alábbi fontossági rangsort kaptuk a mérnökinformatikusoknál:

1. Emelt szintű érettségi programozás feladatán hány pontot értél el | hány pontot tudtál volna elérni?
2. Hányasra értékeled fizikatudásodat?
3. A Code::Blocks fejlesztő környezetet félévkezdést megelőzően…
4. Tanultál már programozási tételeket, nevezetes algoritmusokat?
5. Milyen osztályzatot szereztél matematika érettségin?

6. Milyen adatstruktúrákat használtál?

A 2017-es adatsor az informatikát sokkal jelentősebbnek mutatja, az első hat helyen a programozáshoz szükséges részterületek találhatók:

1. Emelt szintű érettségi programozás feladatán hány pontot értél el | hány pontot tudtál volna elérni?
2. Milyen adatstruktúrákat használtál?
3. Tanultál már programozási tételeket, nevezetes algoritmusokat?
4. Matematikai logikát az szemeszter kezdete előtt…
5. A Code::Blocks fejlesztő környezetet félévkezdést megelőzően…
6. Programozási tételeket korábban…

A matematika érettségi osztályzat a 7., a fizikatudás értéke a 15. helyen található.

Az ELTE IK programtervező informatikus képzésében a programozás tervezést jelent. Korábban a szak neve programtervező matematikus volt. Bár a név változott, a képzés „lelke” maradt. A programozás alapismereteiben az adatoknak mellékes szerepük van, amit lehet egész számmal modelleznek. A tananyag középpontjában az algoritmusok, a módszeres programozás áll. A megtervezett algoritmus alapján készítik el a programot. A nulláról felépített tananyag főleg a programozás algoritmus, logika részére vonatkozik, praktikus okokból kiterjesztve a programkészítésre. Az adatstruktúrákkal csak a legszükségesebb mértékben foglalkoznak. Míg a BME-n stringet készítenek, az ELTE-n a stringet egy kész, minden további nélkül használható adatnak tekintik.

Az ELTE programtervező informatikus képzésén főleg matematikai, illetve az algoritmizálással kapcsolatos ismeretek számítanak:

1. Milyen osztályzatot szereztél matematika érettségin?
2. Milyen adatstruktúrákat használtál?
3. Programozási tételeket korábban…
4. Matematikai logikát az szemeszter kezdete előtt...
5. A Code::Blocks fejlesztő környezetet félévkezdést megelőzően...
6. Emelt szintű érettségi programozás feladatán hány pontot értél el | hány pontot tudtál volna elérni?

Bár a sorrend évről évre változik, a matematika mindig nagyon elöl van, a fizika előtanulmányok minősége nem lényeges. A korreláció figyelése önmagában nem ad teljes képet. Például a nyelvismeretről azt tartjuk, hogy elengedhetetlen előismeret, korrelációval mégsem mu-

tatható ki az előismeret tanulmányokra kifejtett hatása. Ennek az az oka, hogy a vizsgált hallgatók jó nyelvismerettel érkeztek, a jegyüket nem befolyásolta negatív irányba a nyelvismeret hiánya.

7.3

X/3. A programozás félreértelmezése

Látható, hogy a korrelációk sorrendjében mindkét képzésben az 1–6. helyek egyikén szerepel az adatstruktúrákra és a programozási tételekre vonatkozó kérdés. A tantárgyak tematikáinak összevetése alapján [54] is látható, hogy mindkét képzésben a két téma közül az egyik a tantárgy fókusza, erre építi fel a tananyagot; a másik ismeretét elvárja. Azt is láthattuk, hogy a különbség nem csak a képzési tematikában, hanem gondolkodásmódban is megnyilvánul. Sőt, a programírási szokásokat vizsgáló kutatásból [59] az is kiderül, hogy az informatikatanárok a programtervezői szemléletű programozást tanítják.

Az egyes képzéseken gyakran találkozhatunk olyan hallgatóval, aki előzetesen a másik típusú programozást tanulta, másként gondolkodik a programozásról. Az oktatók ezekről a hallgatókról azt látják, hogy valamennyire tudnak programozni, de rosszul. Rosszabb esetben a hallgató meg van arról győződve, hogy ő tud programozni és nem érti, miért nem felel meg az oktatónak úgy, ahogy csinálja; nem érti, hogy mi a feladat.

- A BME-s „problémás” hallgató szemtelen, a C helyett C++ nyelven szeretne programozni, mert ott van string, és vector, nem érti miért kell a pointerekkel foglalkozni. „Persze megbukik a ZH-n, mert a csillagokat összevissza írogatja”.
- Az ELTE-s „problémás” hallgató laza, nem rajzol struktogramot (azt sem tudja, hogy mi az), vagy ha mégis, az elágazásba beleírja a break-et. „Persze megbukik a ZH-n, mert nem érti meg, hogy mi a feladat, nem jön rá, melyik tétel kellene. Kavar összevissza.”

Jellemző, hogy személyes hozzáállás problémájaként detektálják az oktatók a problémát, mert a hallgató vitatkozik, ahelyett, hogy úgy oldaná meg a feladatot, ahogy elvárják tőle. Tapasztalatom az, hogy a személyes viták és az ellenállás háttérben egy tanult, bevált gondolkodási mód áll, amit utasítással próbál az oktató felülírni.

Két speciális, de valószínűleg jellemző példa:

> Kezdve magammal… Három év BME-s óralátogatás, elemzés, oktatás után is nehéz olvasni a String adattípussal kapcsolatos példákat. Van az a string, amit használok C++-ban és van az, amit én megalkotok – lezárónullával vagy a hossz tárolásával vagy mindkettővel – és operátorokat is definiálok hozzá. (Melyikhez is?). Az első két évben a dolgozatfeladatokat nem értettem. Egy 15 perces dolgozat is lehet 1 óra, ha a specifikációt

próbálom kitalálni, azt keresem, milyen algoritmussal oldható meg a feladat, miközben ezek mind nem kellenek, csak – majdnem mechanikusan – definiálni kellene egy dinamikus tömböt.

Volt egy hallgató, aki a BME-n külsősként tanult, ötöst kapott. Majd érettségizett informatikából és utána az ELTE-n nálam tanulta (elvileg) újra a programozást. A dolgozatfeladat lényege az volt, hogy egy köznapi nyelven megfogalmazott problémához adjon specifikációt. A feladat szövegében egy zárójeles utalást az adatok számára nem vett észre, ezért nem értette a feladatot, nem tudott specifikációt írni. Sok időt veszített a vélt információhiány miatt, mert nem mert dönteni, megírni valahogy a megoldást. Inkább semmit sem írt, minthogy olyan megoldást írjon, amit később javítani kell.

Egy tipikus példa:

Az alábbi kód az programtervező informatikus képzés (*pt*) beadandó feladataként készült, de a mérnökinformatikus képzésen (*mi*) beadandó feladat specifikációjának felel meg (ott nem C++, hanem C kódot kellene írni).

```
#include <iostream>
using namespace std;
struct helyseg {
 int* madarak;
};
```

A struct és pointer használata *mi*-n feladat, *pt*-n statikus tömb és cons int maxN használata javasolt.

```
int main()
{
 int N = 0, M = 0;
 cin >> M >> N;

 helyseg* helysegek = new helyseg[M];

 for (int i = 0; i < M; i++)
  helysegek[i].madarak = new int[N];
 for (int i = 0; i < M; i++)
  for (int j = 0; j < N; j++)
   cin >> helysegek[i].madarak[j];
```

Memória foglalás a *mi*-n feladat, a specifikációban elvárt a pointerre mutató pointer (táblázat pointerrel).

Adattípus indirekció *pt*-n nem feladat, struktúra, tömb definiálása statikus megvalósítása a feladat

```
 bool vege = false;

 for (int i = 0; i < N; i++){
  int j = 0;
  for (; j < M; j++)
   if (helysegek[j].madarak[i] != 0){
   }
   else break;
  if (j == M){
   cout << i + 1;
   vege = true;
   break;
  }
 }
```

A kód *pt*-n az NSD-ben elkészített algoritmus implementációja. Nem fogadható el a hiányos for-ciklus és a break; és nem ajánlott az elágazásban az üres igaz-ág. A megoldás értéke: 0

A kód *mi*-n a gondolkodás implementációja. A megoldás nem szép, de jó. A megoldás értéke a feladat megoldásának fokától függ.

```
if (!vege)
 for (int i = 0; i < N; i++){
  int j = 0;
  for (; j < M; j++)
   if (helysegek[j].madarak[i] != 0)
    break;
  if (j != M)
   break;
  if (i == N - 1){
   cout << 0;
   vege = true;
  }
 }
```

A megoldásból látszik, a gondolkodási séma követése. Ez a gondolkodási séma alkalmatlan az összetettebb algoritmusok megvalósítására, illetve olvashatatlan kódot eredményez. A *pt* feladat specifikációjának a megoldás nem felel meg.

```
for (int i = 0; i < M; i++)
 delete[] helysegek[i].madarak;
delete[] helysegek;
return 0;
}
```

A memória felszabadítása rendben, *mi* feladat specifikációnak a megoldás megfelel.

Átéltem „hallgatóként” és oktatóként, hogy milyen az, amikor azt hiszed, hogy tudsz programozni, és mégsem érted, hogy mit várnak el tőled programozás címén. Átéltem, illetve láttam a frusztrációt mindkét oldalon. Azóta mindkét oldalon több „szemtelen”, „laza” hallgatóval sikerült megértetnem, hogy a tantárgyon belül a programozást hogyan értelmezzük.

A közoktatásban tanító kollégák is rendszeresen találkoznak a programozni tudás különböző értelmezéseivel. Kezdve a blokknyelven programozóval, aki nem érti, miért kell szövegalapú nyelven is megtanulnia programot készíteni. Gyakran az IoT fejlesztő sem érti, miért kell még többet tudnia programozásból. Lényegében állandó tanulásért vívott harcot eredményezhet egy-egy „ismerőstől programozást” tanuló diák, aki elágazás helyett a try-catch kivételkezelést alkalmazza; aki a vezérlési szerkezetekhez egyenrangú elemként használja a `goto; break;` utasításokat, vagy aki csak egy adott nyelven hajlandó kódot írni.

A programozás hatékony oktatásához – ahogy az informatika hatékony oktatásához is – az oktatónak pontosan kell tudnia, hogy a teljes tudáshalmaz mely részét tanítja és ez milyen viszonyban van a többi tudományterülettel. A jó tanár egyik ismérve, hogy a hallgatónak el tudja magyarázni, hogy éppen mit is tanul, meg tudja értetni vele, hogy a jelenlegi tudása mennyit ér és a tananyag elsajátításával milyen irányba halad, mi az, amiben később kell fejlődni [28, 142].

7.3

X/4. „Nulláról induló tananyag”

Az egyetemi kutatásom során talán ezzel a kifejezéssel találkoztam legtöbbször. Az oktatók általában úgy érzik, hogy a tananyagot nulláról kell elkezdeni felépíteni. A programozás oktatása módszerének alapkérdése, hogy ennek a felépítésnek mi a fókusza. Szlávi, Zsakó és Papp-Varga – a programozás oktatásának módszereiről írt tanulmányban [149, 150] – különböző módszereket ad a programozás oktatására. Az adott módszereket a programozáshoz szükséges

ismeretek fókusza alapján definiálnak, jellemeznek. Vizsgálják, az egyes módszerek hatékonyságát különböző korcsoportok, oktatási környezetek esetén.

Minden, a tanulmányban leírt módszer „nulláról" építi fel a programozást, csak a korosztály és a környezeti feltételek utalnak arra, hogy valamilyen előzetes tudásra szükség lenne. (Most már nem tudom, hányadszor írom le, hogy „felépíti".) A tanítás során a megtanulandó ismeretek felépítése erősen utal a deduktív gondolkodási[61] mód preferálására. Az egyetem szakmai képzést ad, ahol a törvényszerűségek alapján fejlesztjük ismereteinket, hozunk létre új tudást. A megismerés természetes módja azonban az induktív gondolkodás, az egyedi jelenségekre magyarázatok keresése. Ilyenkor több egyedi jelenségből valószínűsítünk szabályokat. (Mostanában a mesterséges intelligencia önálló tanulása ezen alapul, míg a korábbi MI-k a deduktív szabályokat alkalmazzák. Az élmény alapú tanulás is az induktív gondolkodást helyezi előtérbe.)

Az ELTE IK *Programozás alapismeretek* tantárgya algoritmus-orientált felépítésű, deduktív gondolkodási módot követ, az elemi algoritmusokból építi fel az összetettebbeket. A BME VIK *Programozás alapjai* tárgy elsősorban adat-orientált (hosszabb távon objektum-orientált) felépítésű, szintén deduktívan, az elemi adatokból építkezik a bonyolultabb felé.

A belépési ismereteket felmérő kérdőívvel a matematika, fizika, idegennyelv ismeretének szerepe mellett a programozás ismeretek meglétét, jellegét is mértük. A kérdések több, mint kétharmada a programozással kapcsolatos képességek és készségek részleteire kérdez rá. Közülük több az induktív gondolkodással elsajátított, programozáshoz szükséges készségeket is megpróbálja feltérképezni: a válaszlehetőségek között egy lehetőség a „hallottam róla, de még nem próbáltam".

A válaszok elemzése azt mutatja, hogy azokban a csoportokban – például informatikatanárképzés – ahol az előzetes programozástanulás nem jellemző, más ismeretekre, más tapasztalatokra lehet építeni a programozástanítást. Tanárképzésben a hozott táblázatkezelési ismeretek minősége korrelál legerősebben a tantárgyban elért eredménnyel, de a korábbi évek fizikára építő oktatása is ezért működött az akkoriban felvett hallgatók esetén. A korábbi ismeretekre építő ismeretszerzés alapfeltétele, hogy legyenek korábbi ismeretek. Ezért a táblázatkezelés alapos ismerete jó tapasztalat az adatok, az adattípusok, a hivatkozások fogalmának kialakításához, a fizika feladatmegoldási rutin jó tapasztalat a problémamegoldásra (modellezésre, a bemenet-feldolgozás-kimenet specifikációra).

[61] Az induktív és deduktív ismeretszerzésről egy leírás: http://mmi.elte.hu/szabadbolcseszet/mmi.elte.hu/szabadbolcseszet/index3cd9.html?option=com_tanelem&id_tanelem=475&tip=0 [2021.10.30]

A bemeneti ismereteket feltérképező kérdőív első kérdése: Tanultál programozni? A pozitív válaszok a tanulás kereteire mérnek: iskolai vagy egyéb módon. Akik tanultak programozni, azok – a programozás valamelyik értelmezése alapján – már ismerhetik a tananyagot, számukra a „nulláról induló tananyagfelépítés" egy kibővítéssel teljessé tett rendszerezés. Akik nem tanultak programozni, azoknál kérdéses, hogy a korábbi tapasztalataik mire lesznek elegendők, képesek lesznek-e elszórtan, hiányosan meglevő ismereteiket összeszedni és komplex tudásként alkalmazni. A válaszoknak a féléves eredménnyel való összevetése (38. ábra) megmutatja a tárgy teljesítésének esélyeit különböző előképzettségek mellett.

- A kérdőíves felmérésről általánosan elmondható, hogy a válaszolók eredményei jobbak, mint a teljes évfolyamra számított eredmény. Vélhetően a tudatosabban tanulók, a szorgalmasabbak, a lelkiismeretesebbek töltötték ki inkább a kérdőívet és ezek a tulajdonságok a tanulmányi eredményben is tükröződnek. Összehasonlításképp: a BME mindhárom évében a 0v1 (nem teljesítette vagy elégtelen) értéket a hallgatók közel 30%-a érte el. A felmérésben résztvevők között a 38. ábra első oszlopában látható 19%, 21%, 26% nem jelent romlást. Valójában 2015-ben a későbbi kitöltés miatt „kiestek" a korai lemorzsolódók, ami csökkentette a százalékos arányt; a 2017-ben a kitöltők száma 36%-kal, az évfolyam létszámánál nagyobb arányban nőtt, azaz a minta szélesebb spektrumot fogott át.
- A 2018-as ELTE felmérés az oktatói motiválás elmaradása miatt elég kis létszámú (96 fő ~ 15%), a többi minta 30-50%-os. A kitöltés önkéntes volt.
- Azoknak, akik nem szereztek aláírást a tárgyból, az eredményét 0 értékkel jelöltem, így a 0v1 a nem teljesítőket vagy bukottakat jellemzi. A tudás minősítése szempontjából az elégségesek száma mindkét egyetemen olyan kicsi, hogy érdemes a közepessel együtt kezelni. Hasonlóan a jeles és jó között alig van tudásbéli eltérés. Ezért ezeket az értékeket is egyben vizsgáltam, 2v3 és 4v5.
- A válaszokból kiderül, hogy a válaszadók 13–22%-a nem tanult korábban programozni. Valószínűleg a valóságban ez az érték nagyobb. A BME-n ez az arányszám csökkenő, míg az ELTE-n növekvő. Ez betudható annak az informális hírnek, hogy a BME a legnehezebb, könnyű megbukni, ezért javasolt az előzetes programozástanulás. Ezzel szemben az ELTE-n a mintában benne vannak az esti és tanárszakosok is, ahol a felvételi is teljesen más határokkal történik, esti képzésen a jelentkezés és a részvétel motivációja is teljesen más.

BME						2015	ELTE					
0v1 (db)	2v3 (db)	4v5 (db)	Jegyek átlaga	Válaszok eloszlása	0v1 aránya							
11	9	11	2,65	13%	35%	Nem.						
10	9	44	3,76	26%	16%	Igen, egyénileg, internetről, baráttól... tanultam.						
17	13	28	2,98	24%	29%	Igen, tanítottak az iskolában.						
8	13	73	4,04	38%	9%	Igen, tanítottak az iskolában és egyénileg... is tanultam.						
46	44	156		↓100%		Oszlopösszeg						
19%	18%	63%	→100%			Jegyek eloszlása						
0v1 (db)	2v3 (db)	4v5 (db)	Jegyek átlaga	Válaszok eloszlása	0v1 aránya	2016	0v1 (db)	2v3 (db)	4v5 (db)	Jegyek átlaga	Válaszok eloszlása	0v1 aránya
13	12	14	2,72	16%	33%	Nem.	17	14	26	2,81	**21%**	30%
13	7	36	3,55	24%	23%	Igen, egyénileg, internetről, baráttól... tanultam.	12	7	30	3,27	18%	24%
14	18	28	3,18	25%	23%	Igen, tanítottak az iskolában.	10	22	47	3,58	29%	13%
9	16	58	4,00	35%	11%	Igen, tanítottak az iskolában és egyénileg... is tanultam.	13	15	64	3,77	33%	14%
49	53	136		↓100%		Oszlopösszeg	52	58	167		↓100%	
21%	22%	57%	→100%			Jegyek eloszlása	19%	21%	60%	→100%		
0v1 (db)	2v3 (db)	4v5 (db)	Jegyek átlaga	Válaszok eloszlása	0v1 aránya	2017	0v1 (db)	2v3 (db)	4v5 (db)	Jegyek átlaga	Válaszok eloszlása	0v1 aránya
31	13	23	**2,27**	**21%**	**46%**	Nem.	20	12	15	**2,34**	**22%**	**43%**
14	15	45	3,50	23%	19%	Igen, egyénileg, internetről, baráttól... tanultam.	14	8	24	3,13	21%	30%
28	15	32	2,67	23%	37%	Igen, tanítottak az iskolában.	16	7	35	3,21	27%	28%
11	17	80	3,94	33%	10%	Igen, tanítottak az iskolában és egyénileg... is tanultam.	8	6	50	3,89	30%	13%
84	60	180		↓100%		Oszlopösszeg	58	33	124		↓100%	
26%	19%	56%	→100%			Jegyek eloszlása	27%	15%	58%	→100%		
						2018	0v1 (db)	2v3 (db)	4v5 (db)	Jegyek átlaga	Válaszok eloszlása	0v1 aránya
						Nem.	7	3	5	**1,87**	**16%**	**47%**
						Igen, egyénileg, internetről, baráttól... tanultam.	8	1	18	3,33	28%	30%
						Igen, tanítottak az iskolában.	3	2	16	3,95	22%	14%
						Igen, tanítottak az iskolában és egyénileg... is tanultam.	8	3	22	3,42	34%	24%
						Oszlopösszeg	26	9	61		↓100%	
						Jegyek eloszlása	27%	9%	64%	→100%		

38. ábra: Belépési kérdőív „Tanultál programozni?” kérdésére adott válaszok és a programozás tantárgyban elért eredmények összehasonlítása

- Figyelmeztető jelenség, hogy a válaszadók között azok, akik iskolában is tanultak programozást, csökkenő arányt képviselnek. Ez a 3. és 4. százalékos adat összege, a BME-n 62%, 60%, 56%; az ELTE-n egy évvel későbbi adatokból 62%, 57%, 56%. A 2017-es adat az első 2013-as NAT szerint tanuló évfolyam, ahol a gimnáziumokban lecsökkent informatikaóraszám miatt csak fakultáción lehet valamennyire felkészülni az érettségire, de sok helyen ez a középszintű érettségire történő felkészítést jelenti. Az összesből minden esetben nagyobb az iskolán kívül is tanulók száma.
- Az előképzettség hatását mutatja az adott választ adók jegyeinek átlaga.
  - Minden esetben messze a legrosszabb azoknak az eredménye, akik nem tanultak korábban programozást. Itt a bukottak száma 30–46%. Nagyon szembetűnő, hogy idővel, 2017-ben jelentősen romlik ez az arány mindkét egyetemen. Ennek valószínűleg az az oka, hogy nem csak programozást, de a hozzá szükséges készségeket sem tanulták középiskolában a hallgatók.
  - Azoknak, akik iskolai és egyéni úton is tanultak programozást az eredménye jelentősen jobb, 4-es körüli érték. Itt a bukások aránya 10% körül van, de ez valószínűleg nem a tudás elégtelenségéből fakad. Oka az egyes konkrét eseteket tekintve a már korábban említett programozás mibenlétének félreértelmezése, illetve a képzési körülményekből adódó mentális problémák.
  - Figyelemre méltó, hogy az iskolában, illetve egyénileg tanulók közül a BME-n az egyénileg tanulók eredménye lesz jobb, míg az ELTE-n az iskolában felkészülőké. Ezt leginkább az magyarázza, hogy a közoktatásban oktató tanárok az ELTE-n tanított módszeres programozás paradigmáját készítik elő, ezért akik közoktatásban tanulnak programozni, azok nem fogják félreérteni a követelményeket az ELTE-n, viszont nehézséget okozhat számukra átállni a műszaki szemléletre a BME-n.
- A BME VIK és az ELTE IK a saját területük magyar top képzésének számítanak. A vizsgált csoportok nagyságrendileg azonosak voltak, 500–650 hallgató képzését végzik az egyes tantárgyak keretében. Azonban a BME VIK-en a vizsgált képzés csak a magyar mérnökinformatikus szakot jelenti, amiben az iMSc (a legjobbak tehetséggondozó képzése) jelent más megközelítést, míg az ELTE IK Programozás tantárgyát több szak hallgatói tanulják. A vizsgált csoport magába foglalja a programtervező informatikus nappali és esti képzését, az informatikatanárok nappali és kiegészítőszakos levelező képzését, és a programtervező informatikus (fejlesztő) felsőfokú szakképzést is. A BME VIK-en ezzel

a megközelítéssel a német nyelvű képzés és a villamosmérnök képzés hallgatóit, a hallgatók *Programozás alapjai* tárgyban elért sikereit is vizsgálnom kellett volna és a listát mostanra az üzemmérnök-informatikus képzés vizsgálatával is kibővíthetném. Az ELTE IK-n vizsgált csoport heterogenitása miatt az átlagok mellett egyes kérdésekben a belső tagozódást is figyelembe kell venni.

- Az ELTE 2018-as évfolyama nagyon kis létszámban vett részt a felmérésben (96 fő), a *Programozás* tárgy sikerességének vizsgálata az egyes csoportokra is – ahol érdemi mennyiségű adat született – megvizsgáltam. A 2016-os mérést a programtervező informatikus és az informatikatanár csoportokra külön is kiértékeltem, a tapasztalatokról a [148] cikkünkben írtunk. A 2018-as mérés alcsoportokra értékelését a tanárokra nem végeztem el, mert nagyon kevés (2) tanár töltötte ki a kérdőívet. Az ő eredményük (egyik kimaradt, a másik megbukott) és az évfolyam eredménye is lehangoló: több, mint 50% kibukott. Igaz, a többiek átlaga 3,6

A programtervező informatikus nappali, esti és fejlesztő „tagozatai” külön vizsgálata több szempontból tanulságos. (39. ábra)

- Az alacsony kitöltési arány a nappali tagozat jellemzője, viszont az esti és fejlesztő tagozaton a gyengékre nem mondhatjuk, hogy lusták voltak kitölteni.
- Az egyes csoportokra a hozott matematika érettségi jegyet is érdemes figyelni, mivel itt a legfontosabb előismeretek ebbe a tárgyba tartoznak.
- Az esti tagozat programozás eredménye a válaszadók és a teljes csoportra nézve is két pólusú: vagy bukik vagy jó. A fejlesztők közül körülbelül ugyanakkora a bukók aránya, de ott a közepesen (2v3) teljesítők száma jelentősebb, mint a jóké. (Innen nehéz lesz továbblépni.) A nappali tagozat kétharmada jó, csak 20%-nak nem sikerült a követelményeket teljesíteni.
- A hozott matematikajegynél az érettségi középszintű jelest is jó-nak számoltam, mert az emelt szintű tudást szeretné elvárni az egyetem. Ezt figyelembe véve elmondhatjuk, hogy a válaszadók körében fejlesztő tagozaton nem volt emelt szintű jeles érettségivel érkező. A nappali tagozaton viszont nem volt sem elégséges, sem közepes.
- Van olyan hallgató, aki elégséges matematika érettségit követően sikeresen teljesítette a tárgyat, közepes matematikajegy a tárgy teljesítésére csak 50–60%-os eséllyel elegendő. Esti tagozaton a matematika jegy egy régen meglévő tudást mér, a „jó”, illetve középszintű jeles ebben a körben (60%-os bukási arány) nem segíti a teljesítést.

- Esti tagozaton nem lehet megtanulni programozni. Aki előtte nem tanult vagy nem tudja eléggé, az esténként, éjjel, munka mellett a nappalis követelményeket csak nagyon nehezen tudja teljesíteni. A bukás 60% fölött van. Itt kell megjegyeznem, hogy esti tagozaton kevesebb kontaktóra van. Az eredmény azt is mutatja, hogy csak elméletben lehet tetszőlegesen terhelni a hallgatót, a kivitelezés nem lehetséges. Annak ellenére, hogy a számonkérésnél gyengítettek – egy picit – a követelmények, a hiányzó tudás és gyakorlat nem kicsit vagy lineárisan lesz kevesebb, hanem szakadékokat képez.
- A fejlesztőknél nem számít, hogy tanult-e korábban programozást. A 7 nem és 6 igen válaszból 2-2 bukott (kb. 30%). Bár a minta nagyon kicsi, ha az arányok a teljes tagozaton ehhez hasonlók, az azt jelentheti, hogy aki nem tanult, azért bukik, aki viszont előre tanult, az egész mást tanult, nem tudja hasznosítani a korábbi ismereteit. Ezt érdemes alaposabban elemezni.
- Nappali tagozaton nagyon kevésnek tűnik azok száma, akik nem tanultak a megelőző időszakban programozást. Valószínűleg ebből a szempontból a válaszadók nem jellemzik a teljes évfolyamot. A kitöltők eredményessége sokkal jobb, mint a teljes évfolyam átlaga. Így vélhetően sok gyengén teljesítő nem töltötte ki a kérdőívet. Ha ezt a motiváltság hiánya okozta, akkor a kérdőívet nem kitöltők között arányaiban több lehet a kevésbé felkészült, programozást korábban nem tanuló hallgatók száma is.

7.3.1

2018 ELTE Matematika (középszintű 5 =>4) szakok szerinti bontásban — **2018 ELTE Programozás szakok szerinti bontásban**

Mat	0v1 (db)	2v3 (db)	4v5 (db)	Jegyek átlaga	Mat. jegy eloszlása	0v1 aránya	programtervező informatikus (esti)	0v1 (db)	2v3 (db)	4v5 (db)	Jegyek átlaga	Válaszok eloszlása	0v1 aránya
2	1	0	0	0,00	4%	100%	Nem.	3	0	2	1,60	19%	**60%**
3	2	0	2	2,25	16%	50%	Igen, egyénileg, internetről, baráttól... tanultam.	7	0	4	1,73	42%	**64%**
4	**9**	1	5	1,80	60%	**60%**	Igen, tanítottak az iskolában.	**1**	0	1	2,00	8%	50%
5	2	0	3	2,60	20%	40%	Igen, tanítottak az iskolában és egyénileg... is tanultam.	4	1	3	2,38	31%	50%
	14	1	10		↓100%		Oszlopösszeg	15	1	10		↓100%	
	56%	4%	40%	→100%			Jegyek eloszlása	58%	4%	38%	→100%		

Mat	0v1 (db)	2v3 (db)	4v5 (db)	Jegyek átlaga	Mat. jegy eloszlása	0v1 aránya	programtervező informatikus (fejlesztő) felsőoktatási szakképzés	0v1 (db)	2v3 (db)	4v5 (db)	Jegyek átlaga	Válaszok eloszlása	0v1 aránya
2	0	1	0	2,00	**8%**	0%	Nem.	2	3	2	2,29	**54%**	29%
3	2	3	0	1,80	**38%**	**40%**	Igen, egyénileg, internetről, baráttól... tanultam.	1	1	0	2,00	15%	50%
4	2	2	3	2,86	54%	29%	Igen, tanítottak az iskolában.	0	1	1	3,50	15%	0%
5	0	0	0	-	**0%**	-	Igen, tanítottak az iskolában és egyénileg... is tanultam.	1	1	0	2,00	15%	50%
	4	6	3		↓100%		Oszlopösszeg	4	6	3		↓100%	
	31%	46%	23%	→100%			Jegyek eloszlása	31%	46%	23%	→100%		

Mat	0v1 (db)	2v3 (db)	4v5 (db)	Jegyek átlaga	Mat. jegy eloszlása	0v1 aránya	programtervező informatikus (nappali)	0v1 (db)	2v3 (db)	4v5 (db)	Jegyek átlaga	Válaszok eloszlása	0v1 aránya
2	0	0	0	-	**0%**	-	Nem.	1	0	1	2,00	4%	**50%**
3	0	0	0	-	0%	-	Igen, egyénileg, internetről, baráttól... tanultam.	0	0	14	4,79	26%	0%
4	1	1	19	4,10	40%	5%	Igen, tanítottak az iskolában.	1	1	14	4,44	30%	6%
5	3	1	28	4,44	60%	9%	Igen, tanítottak az iskolában és egyénileg... is tanultam.	2	1	19	4,09	41%	9%
	4	2	47		↓100%		Oszlopösszeg	4	2	48		↓100%	
	8%	4%	89%	→100%			Jegyek eloszlása	7%	4%	89%	→100%		

39. ábra: Belépési kérdőív Matematika tudás és „Tanultál-e programozni?” kérdésre adott válaszok és a programozás tantárgyban elért eredmény összehasonlítása

X/5. Kódolás tanítása a BME VIK-en

A *Haladási napló* adatainak elemzésekor [146, 147] feltűnt, hogy a „Debug” szempontra adott értékelések nem korrelálnak az elért eredménnyel.

Debug – tudásellenőrzés

0. Nem tudom, mit jelent a "debugolás".
1. Debugolással nehéz a hibák megtalálása.
2. A debug hibajelzései, lépésenkénti futtatás segít a program működésének megértésében.
3. A hibák megtalálása könnyebb Debug módban, mintha át kellene olvasnom.
4. A programkód írása során a szintaktikát és szemantikát is a fejlesztőkörnyezet adta eszközzel ellenőrzöm.
5. A kódírás része a kód mentése, ellenőrzése. Akkor is gyakran végzem, ha nem figyelek rá.

A kérdések valójában arról szólnak, hogy a fejlesztőkörnyezet adta hibakeresési és javítási lehetőséget mennyire használja a hallgató, része-e a munkájának a gyakori ellenőrzés, javítás. Az összes ZH pontszámmal és a végső eredménnyel is alacsony korrelációt mutattak a válaszok, ami azt jelenti, hogy a mérések az adott készséget nem mérték. A „Debug” szempontra adott értékelés úgy korrelál a végeredménnyel, mint a „Bitműveletek” és a „Rekurzió”. Egyik sem szerepelt a ZH-kban. Azonban a bitműveletek és a rekurzió esetén ez szándékos, a két téma ismertető jelleggel volt beépítve a tananyagba, míg a programozás alapjaihoz a tesztelés, hibakeresés hozzátartozik.

Ezzel párhuzamosan, a belépéskor felvett kérdőív gépírásra vonatkozó kérdései az eredménnyel negatív korrelációt jeleztek.

Gépírás 1. Melyik igaz leginkább?

- Tudok 10 ujjal vakon gépírni (értéke: 5).
- Nem 10, de legalább 5 ujjal tudok gépírni (értéke: 4).
- 2–4 ujjal tudok gépelni (értéke: 3).
- Eddig nem szereztem gyakorlatot gépírásból, többnyire képernyő-billentyűzetet használtam, vagy azt sem (értéke: 2).

A másik kérdés arra vonatkozott, hogy a gépelés a gondolkodással milyen viszonyban van, tud-e a hallgató gépelés közben gondolkodni, vagy akkor csak arra figyel, hogy megtalálja a billentyűt. E két kérdés negatív korrelációja az eredményekkel azt mutatja, hogy aki könnyedén,

természetes kifejezőeszközként kezeli a billentyűzetet, az kevésbé tud teljesíteni a dolgozatokon, aki kevésbé tud gépelni, annak jobban sikerülnek a dolgozatok.

Az eredmények hátterében az áll, hogy a BME *Programozás alapjai* tantárgyban egy kivétellel minden számonkérés papír alapon történik. A kivétel a beadandó házi feladat, aminél a gépelés módjáról és idejéről semmit nem tudunk, a tesztelés, hibajavítás pedig csak kritikus esetekben – nem sikerült megtalálni a hibát és ezért nem tudta leadni – hat ki az eredményre.

A papírra írt kód az előadás szemléletmódjával ellentétes. Az előadó az előadás közben élesben ír, futtat és tesztel kódot. Ez az egyik, nagyon hangsúlyos oka annak, hogy az előadásait teltházzal tartja még november végén is. Ő az, aki megmutatja, hogyan kell programot írni: miközben mondja, hogy mit csináljon a program, írja a kódot. „Kell egy egész" közben írja, hogy „int a";. Azt is láthatják a hallgatók, hogy hogyan fejlődik a program, az egymás elé-után, egymásba elrendezett kódrészletek hogyan keletkeznek. A táblára csak ritkán ír. Például a függvény tanításakor, de akkor demonstrálja, hogy nem lineárisan írjuk a kódot, hanem a gondolkodás sorrendjében: függvény neve, paraméterei, majd az elejére a visszatérési érték típusa.

A papíron kódolással a problémák a gyakorlatokon kezdődnek. Elvileg nem arról kellene szóljon a gyakorlat, hogy az előadó magyaráz. Mégis, ha a maximum egy kivetítő, amin a gyakorlatvezető ki tudja vetíteni saját gépéről a megoldást, akkor a hallgatók aktivizálásának a lehetősége a wireless billentyűzet átadása a hallgató keze alá. Ezt a módszert csak egyetlen esetben láttam, sőt, egyes gyakorlatokon a kivetítő sem természetes tartozék. Az alapvetően tábla-kréta és laptop-kivetítő eszközökkel oktató tanárok és papírra jegyzetelő hallgatók a gyakorlatok során feladatokat oldanak meg, megbeszélik a megoldást. Jó lenne, ha a hallgatók önállóan oldanák meg a feladatokat, de abba nehéz beleírni a javítást. Különösen nehéz egy feladat továbbfejlesztéseit beleírni, abból tanulni. Emiatt általános munkamódszer az, hogy az oktató a hallgatók bevonásával megoldja a feladatokat. Frontálisan. Utána a gyakorlat jegyzetét a hallgató megkapja, hogy házi feladatként rekonstruálni tudja. Az álmosság, alacsony motiváltság néhány hét elteltével a hallgatók jelentős részénél bambán nézéssé alakul és a gyakorlatvezető átlagosan 3-4 hallgató társaságában oldja meg a feladatokat. Az *Ellenőrző feladat* előírásszerinti íratása és javítása épp azért jelentős, mert a hallgatók aktivitását, önálló és csoportmunkáját motiválja. Azonban a gyakorlatvezetők aggályukat fogalmazták meg, mert lassú, sok időt vesz igénybe, nem tudnak „haladni a tananyaggal". A haladási kényszer abban is megnyilvánul, hogy a gyakorlatok alatt minél több helyes megoldást a hallgatóknak is le kell írni. Gyakorolni kell a papírra kódolást, mert a ZH-kat így írják.

Az otthoni felkészüléshez a feladatok gépen megoldása, tesztelése a javasolt tanulási módszer, azonban a ZH-ra készüléshez mindenképp gyakorolni kell a megoldások papírra írását is.

Példa: Kipróbáltam a tanulást…

Bár húszéves gyakorlatom van programozásból, lényegében újra tanultam. A tervezés, módszeres programozás helyett a program funkcionalitásának fejlesztése az, amit követni érdemes:

A feladatot először végigolvasom, hogy átfogó képet kapjak róla, majd részenként megvalósítom. Létrehozom az adatot, használom, törlöm; tesztelem, hogy jó-e; elé megírom a feladatszerinti első felhasználást vagy műveletet; tesztelem… A program írása során a feladatot többször elolvasom, mint egy check-listet, ellenőrzöm, hogy a programom a kiírás minden elvárását teljesíti [32].

Papíron írt kód esetén jellemzően először egy műveletet, függvényt kell megírni, majd ezt követően azt a főprogramrészletet, amiben ezt felhasználom. Ahhoz, hogy ez jó legyen, a feladatot az írás megkezdése előtt teljes egészében át kell látni, meg kell érteni. Ha valami miatt elnézek valamit vagy túlbonyolított absztrakciót találok ki, akkor a javítás valószínűleg teljes újraírást jelent – ami a ZH-n nem fér bele az időbe. Ha úgy gondolom, hogy tudom, mit fogok írni, a teljes megoldás vázlata, belső összefüggései a fejemben megvannak, akkor kezdem a kívánt sorrendben megírni a kódot.

Míg a gépen írt kód esetén top-down fejlesztést végzek, a papíron bottom-up a leírás sorrendje. Sőt, papíron a blokkok szervezése helyett lineárisan kell írni. Ez azt jelenti, hogy a ZH megírásához az adott feladattípust nem csak megoldani kell tudni, hanem a megoldást be is kell magolni. Sajnos ebből az is következik, hogy a ZH-ra készülés egyik módja a gondolkodás helyett a magolás, ami kezdetben nem okoz gondot, de később nehéz lesz róla átállni.

A kutatásom során több tárgyból vettem részt a gyakorlatokon. *Programozás alapjai 1, 2* gyakorlatokat 2-3-szor ültem végig, a C++11 gyakorlatait is 2-szer jártam végig, Rendszermodellezést végigcsináltam, az Adatbázisokból több gyakorlaton bent voltam. Az első két évben egyetlen papír alapú ZH-t sem írtam meg, a második két évben megpróbáltam megírni a kis ZH-kat, beugrókat a hallgatókkal együtt. Minden esetben arra volt jó a kísérlet, hogy elkeseredjek a bénaságomon. Az esetek többségében a feladatok szövegében volt valami, amit nem értettem pontosan. Volt, hogy egy vessző hiányzott, volt, hogy egy szó volt számomra kétértelmű vagy a helyzet volt ismeretlen. Meg kell állapítanom, hogy sem az autók, sem a szerszámok, sem a Csillagok háborúja, de még a

kávék szakszavai sem egyértelműek számomra. Ennek a problémának a leküzdése gépen írva még belefért volna az időbe, mert elkezdem az egyik értelmezéssel, legfeljebb közben módosítok… de papíron csak ültem felette és igyekeztem törölni az értelmetlen szavakat, hátha az is elég a megoldáshoz, ami marad.

Tépelődés után jött az írás. Sokkal lassabban írok kézzel, mint géppel. Azt hiszem, a csúnya írásom csak kétszerannyi ideig tart, de akkor betűket hagyok ki, szavakat cserélek fel, felismerhetetlenek a betűim. Ha olvashatóan írok, akkor viszont nem tudok közben gondolkodni, sokkal lassabban megy. Például: a scanf(″%d″,&a); leírása közben körülbelül az első 'a'-nál eszembe jut, hogy a másodiknál nem szabad elfelejteni az &-et. A nyitó idézőjelnél „jaj, csak nehogy elfelejtsem zárni", majd ezután tényleg elfelejtem bezárni az idézőjelet. Újraolvasás, kiegészítés – néha a betűket is átírom, a 'd' nehogy 'a'-nak látszódjon. Mire leírom a sort, már újra át kell gondolnom, hogy mit és miért kértem be. Közben észreveszem, hogy fogy az idő, ezért ideges leszek, és még lassabb: „Tudom, hogy tudom, de nem lesz időm leírni, milyen gáz, hogy hibás lesz…"

Pontosan tudom, hogy az önmagunkon végzett kísérlet veszélyeket rejt. Nem állítom, hogy mindenkinek gondot okoz a gépen gyakorlás után papíron ZH-t írni. De az biztos, hogy van, akinek problémát jelent, például nekem. Ismerek olyan hallgatókat, akiknek szintén problémát jelent, olyanokat is, akiknek „van papírja" arról, hogy gépen írhatja a dolgozatokat. Az pedig jelzés értékű, hogy a kis ZH-k kicsi feladatait jól megoldó hallgatók az első nagy ZH-t nem tudják 50%-ra megírni, majd ezt követően nem is próbálkoznak a javítással. Ez a reakció a nem várt, nem javítható problémára utal, ami lehet rossz tanulási módszer, például a magolás, vagy teljesíthetetlen számonkérési forma például a kézírás.

A hallgatók papíron kódolási teljesítményét valószínűleg hátrányosan érinti az is, hogy a ma egyetemista hallgatók a jegyzetfüzet helyett jellemzően mobilt használnak, aki érettségizett informatikából az számítógépen mutatta be tudását. Kérdés, hogy szükség van-e arra, hogy időt, energiát fordítson oktató és hallgató a papíron kódolás megtanulására, gyakorlására. A jövőt, a szakmai elvárásokat tekintve semmi sem mutat arra, hogy bárhol, bármelyik cég, bármilyen számítógépes környezet vagy a mesterséges intelligencia a kézírást preferálná.

Ezután már csak az a kérdés, hogy miért írják papíron a kódot a hallgatók. Ennek két oka van: hagyományosan így alakult és nincs megfelelő infrastruktúra.

7.3.2

X/6. Specifikáció és modellezés az ELTE IK-n

Az ELTE *Programozási alapismeretek*, illetve Programozás tantárgyak előadása a programozásból a specifikációra és a modellezésre, struktogramon megjelenítésre fókuszál. Kód csak a nyelvi megvalósítás bemutatásának mértékéig, illetve a prezentáció jegyzet részében látható. Az előadáson nincs kódolásból demó, a példák a gondolkodásmódra vonatkoznak: hogyan értelmezzünk egy feladatot, hogyan találjuk ki mi a specifikációja, hogyan származtassuk a specifikációból a struktogramot.

A bemeneti ismereteket mérő kérdőív gépírással kapcsolatos kérdések korrelációja az egyes mérésekkel körülbelül 0 (2016-ban [-0,07; 0,06], 2017 és 2018-ban [-0,1; 0,08]). A gépes ZH-knál inkább pozitív, a papíron írtaknál inkább negatív a korreláció értéke. A gépelés technikájára vonatkozó (Gépírás 1.) kérdésre inkább pozitív, a gépírással egyidejű gondolkodás inkább negatív korrelációt mutat a végeredménnyel. Az összehasonlításokat befolyásolja, hogy a 2017-es és a 2018-as évfolyamok a gépes évfolyam ZH mellett papíron is írtak évfolyam ZH-t.

Az ELTE-n a programkódot mindig gépen írják a hallgatók, az értékelés is automatikus, feketedoboz módszerrel történik.

Specifikáció

A specifikáció matematikai – halmazelméleti, logikai – jelölésrendszert használ, amit a sok speciális jel és indexelés miatt általában könnyebb kézzel írni. Problémát jelent, hogy ahhoz, hogy publikálni lehessen a specifikációt – például a program dokumentációjában – ahhoz számítógépen is tudni kellene implementálni, de ennek a speciális tudáselemeit nem tanulják a hallgatók. A kézírásfelismerők csak érintőképernyőn működnek és gyengén kezelik az indexeléseket. A speciális karakterek átírásait, lineáris képletbeviteli módokat meg kellene tanulni – például: i \in \doubleN automatikus javítási beállítások alapján i ∈ ℕ alakra váltanak –, de az indexek kezelése, a többszintes kifejezések megjelenítése itt is nehézkes, az erre specializálódott TeX nyelv nem tananyag. A matematikai egyenletszerkesztők külön beépülő elemként jelennek meg, a használatuk lassú. A matematikai jelölésrendszer célja, hogy kézzel írva, tömören, de pontosan fejezze ki a gondolatot. A számítógépes megvalósítás lassabb.

A specifikáció a bemenet és kimenet meghatározásakor adatdefiniálást jelent. Itt a leírást a tömbök dimenziójának jelzése nehezíti. A legegyszerűbb megoldás a felső index helyett a kalap jel használata. Az előfeltétel és utófeltétel megadásában a tömbelemekre hivatkozás hagyományosan alsóindexszel történik, de programozásban a szögletes zárójel a megszokott. Az utófeltétel lényegében egy deklaratív utasítás, ahol – a számítási műveleteknél, integrálásnál, határértékszámításnál használt jelölésrendszer alapján – a kifejezés alá/fölé kerülnek a paraméterek,

nem ritkán több sorban. A paramétereket a programozásban jellemzően a műveleti jel után zárójelbe írjuk. A specifikáció megadásánál – úgy érzem – a köznyelven megadott feladatot egy matematikai kitérővel fordítjuk le informatikai problémára, de ennek a tudományok közötti kitérőnek nem látszik az értelme, mivel a napjainkban képzett programozók nem matematikusok.

A specifikációhoz megfontolandó a jelölésrendszerben egy funkcionális nyelv jelöléseit használni. Nemcsak azért, mert probléma a kézzel írt dokumentáció használata a programozásban – programokról szóló kommunikációban -– hanem azért is, mert a specifikáció a deklaratív, funkcionális programozás gondolkodásmódját tükrözi. Oktatás szempontjából a specifikáció „helyettesíthető" a feladat táblázatkezelőben történő megoldásával, modellezésével. A matematikai jelölésrendszer elhagyásával, módosításával közelebb kerülne a program funkcionális nyelvű – például Haskel – implementálásához és az adatbáziskezelés modellezéséhez, a sor és oszlopkalkulusok értelmezéséhez.

Másik lehetséges módja a megoldás elkészítésének a felgyorsítására, ha korábbi kifejezések gyűjteményét lehet használni az újabb feladatok megoldásának alapjaként. Ez a távoktatás során jól alkalmazható, de ellentmond a „semmilyen eszköz nem használható" ZH kritériumnak.

Struktogram

A program modellezése Nassi-Shneidermann diagrammal történik, mert ez biztosítja a módszeres programozás környezetét és az eredmény szemléletes. Mivel vizualizálja a megoldást, könnyen érthető. Bár a struktogramra is csak elektronikusan, tudományos munkákban, esetleg a program dokumentációjában van szükség, az elektronikus előállítás egyéni feladat, a képzésben a diagram rajzolását tanítjuk. A rajz elkészítésénél ugyanaz a probléma, mint amit a BME-n tapasztaltam a papíron írt ZH-nál: úgy lehet valamennyire értelmes rajzot készíteni, ha előre fejben megtervezem, átlátom, hogy mit szeretnék lerajzolni. Előre kell tudnom, hogy mekkora helyre lesz szükségem, elfér-e a lapra, belefér-e a szöveg. Az oktatás során a struktogramok rajzolását leszűkítettem néhány sablonra, amelyek csak vonalakat tartalmaztak. Ezeket hozzárendeltem az egyes programozási tételekhez. Az összetett feladatokat lebontottam az alapsablonokra, mert így tervezhetőbb, hogy egy-egy rajzolt egység elfér-e a lapon. Ez a módszer erősíti, hogy a feladatmegoldás során a tervezési egység ne a vezérlési struktúra legyen, hanem a már tanult algoritmusok, de ezzel együtt a kreatív helyett a sablonos megoldások lesznek preferáltak. Lényegében azt tanítom, hogy egy-egy programozási tételhez melyik rajzos sablont használja a hallgató. Miután ezt bemagolta, már gondolkodhat azon, hogy jól töltse ki a rubrikákat.

Több számítógépes megoldást megnéztem. A megoldások egy részénél a diagram elkészítése bottom-up sorrendet igényel (táblázatkezelő, vektorgrafikus rajzoló programok) és gondot okoz az elágazás kezelése, mert vagy a ferdevonalak vagy a teljes feltétel trapéz/háromszög alakjának problémás a megjelenítése. Több ingyenes szerkesztő program elérhető (structorizer, edraw, smartdraw, Code::Blocks NSD diagramszerkesztője), azonban ezeknek a kódnyelve nem teljesen felel meg a tárgyban tanultaknak. Például a structorizer az egyenlőségjel helyett nyíllal jelöli az értékadást. Egy részük azonban a diagramból szöveges kód generálására is alkalmas, de negatívum lehet, hogy a módszeres programozásba nem tartozó vezérlési elemeket is megenged. Ilyen például a Code::Blocksba integrált diagramkészítő eszköz. Ígéretes alternatívának tűnnek a blokknyelvek, de ezekben az elágazás ágai nem egymás mellett, hanem egymás alatt jelennek meg, ami más vizuális hatást ad. Az elrendezés ilyen módosítása ad egy szabadsági fokot az ábrázoláshoz: vízszintesen tetszőleges mértékben bővíthető; de az alternatív programágak egymás alatt elhelyezkedése a programfolyam lehetséges útvonalait nem szemlélteti. A blokknyelvek előnye lehet, hogy a közoktatásban is megjelenik, de komoly zűröket okozhat, hogy az egyes blokknyelvekben eltérő blokkok vannak. Például a Scratch feltételes ciklusa a kilépés feltételét várja el és nem a ciklusmag végrehajtásának a feltételét. Emiatt a keresés tételének tanításánál, az algoritmus szemléltetésénél pont nem azt hangsúlyozza ki, ami a kódban kellene.

Az alulról felfelé tervezés miatt a kézzel rajzolt struktogram valódi tervező munkára alkalmatlan. Az elkészített diagram az algoritmus szemléltetésére – ha a rajz minőségi – alkalmas, de ilyen jellegű felhasználása csak a ZH-kon van. Egyetlen haszna az, hogy az elemi vezérlési struktúrák helyett nagyobb egységek sablonjaiban kell gondolkodnia a hallgatónak. A számítógéppel való kivitelezési lehetőségek között van használható, de ezek jellemzően valamilyen kompromisszumot igényelnek: vagy a gondolkodási készség fejlesztésben vagy a használt nyelvi eszközökben vagy a megjelenésben. Ezzel együtt is, hasznosabbnak gondolom bármelyik eszköz használatát, mert az elkészítés gyakorlásával egy később is felhasználható készséget fejlesztünk. Elképzelhető, hogy néhány év múlva blokknyelv jellegű vagy NSD-nyelv készül, nem a gyerekek programozásra motiválása céljából, hanem a módszeres programtervből fordítás céljával, ugyanakkor csekély esélyt látok a kézzel rajzolt ábrák kóddá alakítására.

7.3.2

X/7. Jelen és jövő

A programozás a „számítógép megtanítása”, kommunikáció a számítógéppel. Programozáskor gondolatainkat a számítógép számára értelmes formában prezentáljuk. A papíron történő programfejlesztés fizikailag nem kapcsolódik a számítógéphez, csak az emberek közötti kommunikációra alkalmas. Programozás tekintetében, a kezdeti időket idézi, amikor még gépidőt kellett kérni, ezért a gondolatainkat előzetesen papíron fejezzük ki, alakítgatjuk, ellenőriztetjük a főnökkel vagy tanárral… hogy mire a géphez „szólhatunk”, már mindent úgy „mondjunk”, ahogy szeretnénk.

Napjainkban a számítógépek a kód szintaktikai és szemantikai ellenőrzésére sokkal alkalmasabbak, mint a tanár. A gépen írt megoldások jelentős részben automatikusan javíthatók [145], a gép – a tanár által felügyelt módon – megoldási, javítási javaslatokat adhat. A tanulás távoktatásos módon is lehet hatékony, az egyéni megoldások megosztásával azt a közösség tudja értékelni, a közösség minden tagja tanulhat belőle.

Az iparban még ennél is géphez kötöttebb a programozás. A programok felhőben készülnek, verziókövetéssel, csoportmunkában. Minden fázis gépen rögzített. Megkérdeztem, mi indokolja a papíron kódolást. Egyrészt, a megszokás. A baby-boom generáció hozzászokott a papíron tervezéshez, nem természetes számukra, hogy kéznél legyen egy gép, miközben gondolkodnak; a tollat vagy ceruzát fogják, esetleg rágják… Másrészt, a pénz. Nincs megfelelő környezet. A hallgatóknak nincs programozói munkakörnyezetük az egyetemen. A programozói munkakörökben alapvetően több monitoron dolgoznak, egy egyetemistának legfeljebb egy gép és hozzá egy közepes méretű monitor jut. Ehhez hozhatja a saját laptopját, de töltésre már nem mindenhol van lehetőség. Harmadrészt, az oktatói környezetet kialakítása, haladást követő mesterséges intelligencia létrehozása idő.

Jellemző, hogy ahol a megszokáson túllépve gépen dolgoznak a hallgatók, ott jutott idő az oktatói környezet fejlesztésére is, van lehetőség a hallgatók munkájának tárolására, oktatói ellenőrzésre. Az egyetemeket is érintő karantén és távoktatás egyszerre hatott mindkét gátló tényező oldására.

7.3.2

X/8. Egyetemi képzési formák, oktatási elvárások

Egyének kiváltságától a tömegképzésig

A vizsgált két képzés hallgatói létszáma évfolyamonként 500–650 fő. A közoktatásban egy évfolyam ennek az ötöde, egy oktatott csoport körülbelül az 1/25-e. Az informatikaoktatás tartalmi megszervezése – a tanterv és a kimeneti követelmények – szintjén, azaz országosan, egy évfolyam 100 000 fő, a vizsgált képzések létszámának 200-szorosa. Az egyetemi képzés régen csak a legjobbaknak volt elérhető, a felvettek aránya néhány százalék volt; 1-1 szak néhány csoportot jelentett. Az elmúlt mintegy 150 évben az egyetemi képzés jelentős változáson ment át. Az ELTE [151] és a BME [152] történetében található adatok jelzik a fejlődés mértékét:

> *ELTE: „Az egyetem hallgatóinak létszáma 1890-re a kiegyezés idejéhez képest 70%-kal nőtt … A diákok számát tekintve a budapesti egyetem a világ legnagyobb egyetemei közé tartozott, például az 1900/1901-es tanévben 5661 hallgatója volt."*
>
> ELTE: „Fél évszázad alatt nyolcszorosára nőtt a hallgatók száma (1914-ben 8185 főre)"[62]
>
> *BME: „Az 1940–50-es évek fordulóján nagy, elsősorban mennyiségi fejlődésen ment át az egyetem. 1952-re 1938-hoz képest a hallgatók létszáma 983-ról 1285-re, az oktatóké 208-ról 979-re nőtt."*
>
> *ELTE: „A hatvanas évek közepére a hallgatói létszám növekedése problémákat okozott. Egyrészt, mert nem járt együtt az oktatói létszám arányos növekedésével, aminek gazdasági fedezetét a kormányzat nem tudta biztosítani – és ez gondot okozott a reformok által bevezetett kiscsoportos oktatási formában."*
>
> *ELTE: „Az oktatás korszerűsítése, átalakítása szinte folyamatosan zajlott 1990-től... A rendszer létrehozása együtt járt a tantervek, oktatási programok teljes átalakításával. A rendszerváltás után a hallgatók létszáma ugrásszerűen megnőtt, az évtized végére elérte a húszezret."*

A publikus statisztikák alapján, az elmúlt 25 évben hatszorosára nőtt a diplomások száma[63], magyar egyetemeken 2019-ben nagyságrendileg 250 000 hallgató tanul és 23 000 oktató dolgozik[64]. Az ELTE hallgatóinak száma körülbelül 27 000 fő, amiből 2500 fő az Informatika Kar

[62] Ez nem a részletes történeti leírásból, hanem az összefoglalóból származik: (https://www.elte.hu/content/az-egyetem-tortenete.t.4?m=18)

[63] https://www.ksh.hu/docs/hun/xftp/idoszaki/mikrocenzus2016/mikrocenzus_2016_4.pdf [2021.10.30]

[64] https://dari.oktatas.hu/fir_stat_pub Szűrés: Év:https://dari.oktatas.hu/fir_stat_pub 2017 Adatlista: 1.1. (stat2017_1_1.xlsx) [2021.10.30]

hallgatója[65] A BME hallgatóinak száma körülbelül 21 000 fő[66], amiből 5000 fő[67] a Villamosmérnöki és Informatikai Kar hallgatója. A BME központi adataiban szereplő kari bontásból azt is megtudjuk, hogy a VIK-en 225 teljes állású oktató van.

Az ELTE-n 150 év alatt 27-szeres lett a hallgatói létszám. Eközben a mai Magyarország területén lakók száma csak körülbelül 1-2-szeresre[68] változott. Ez azt jelenti, hogy 150 éve a lakosság körülbelül 0,5%-a volt diplomás, a közelmúltban diplomát szerzők a korosztályuk 33%-át teszik ki, de ez még a 39%-os EU átlaghoz viszonyítva gyenge[69]. A konkrét számokat csak nagyságrendileg értékelve is látszik, hogy az egyetemi végzettség, a diploma szerepe jelentősen változott. A tudomány és technológia fejlődése a „diploma" jelentését a „legmagasabb végzettség" jelentésről a „szakmai kompetenciával rendelkezik" jelentésre módosította. Ez a módosítás jelentős részben az utóbbi 50 évben zajlott, az információs társadalom kialakulásával együtt, részben ennek hatására.

Oktatói szerep változása

A diploma megszerzése tömegek számára lett cél, ami az egyetemek életét, az ott dolgozók munkáját is jelentősen megváltoztatta. A BME történetében [152] olvashatjuk, 1952-ben 1285 hallgató volt, akiket 979 oktató tanított. Hihetetlen adat… Egy oktatóra 1,3 hallgató jutott. A kivitelezés szempontjából: ha minden oktató hetente 1 alkalommal 2 órát foglalkozik egy 15 fős csoporttal, akkor a 86 csoportot csoportonként 11–12 tanár tanítja, heti 21 órában. Azaz hallgatói részről 15 fős csoportokban, heti 21 tanóra, oktatói részről hetente 2 óra 1 csoportnak. Emellett az oktató egyben kutató is, a szakmájának jeles képviselője. A 2018-as KSH adat szerint[70] egy oktatóra 8,9 hallgató jut. A BME VIK-en az adatok alapján ennél jóval nagyobb ez az arány 5000/225, azaz 22 hallgató jut egy oktatóra. Heti 30 órával, 20 fős csoportokkal számolva a VIK 5000 hallgatójának 250 csoportban 7500 órát kellene megtartani. Ehhez van 225 oktató, így egy oktatónak heti 33 tanórát kellene megtartania (… és mellette kutatnia). Ez így nem megy… a csoportos foglalkozások mellett jelentős az előadások száma. Az első félévben (a tantervi hálók alapján) a BME-n az órák fele, az ELTE-n az órák 1/3-a előadás. Azaz a BME esetén heti 15 órája van 250 csoportnak és heti 15 óra előadása van (300 fős teremmel számolva)

[65] https://www.elte.hu/kozerdeku (2019. márciusi adat, évente frissül) [2021.10.30]
[66] https://www.bme.hu/sites/default/files/csatolmanyok/tenyek_es_adatok_2019.pdf [2021.10.30]
[67] https://www.vik.bme.hu/fmd/539.html (2019. márciusi adat, évente frissül) [2021.10.30]
[68] Becslés alapja: https://hu.wikipedia.org/wiki/Magyarorsz%C3%A1g_n%C3%A9pess%C3%A9ge [2021.10.30] „Nagy Magyarország" területének népességére a kiegyezést követő időre 12–18 millió főt láttam. Az itt megadott 5,3 millió fő a mai Magyarország területére reálisnak tűnik.
[69] https://eduline.hu/felsooktatas/Csokken_a_diplomasok_aranya_Magyarorszagon__ECTIFY [2021.10.30]
[70] 2.2.5 Az oktatás környezete, feltételei (2003-2018) https://www.ksh.hu/thm/2/indi2_2_5.html [2021.10.30]. Az egy oktatóra jutó hallgatók száma a felsőfokú oktatásban, fő (2018)

körülbelül 17 csoportnak. Ez az oktatók részéről heti 17-18 tanórát jelent. Az oktatást megfelelő oktatói irányítás mellett segíthetik demonstrátorok. Ők általában csak 1 évvel járnak az oktatott tárgy felett, de – amennyiben a tárgyat magas szinten teljesítették és van adottságuk társaik mentorálására – a demonstrátori szerep még hasznos is számukra: a tanítás a legjobb tanulás. Erre csak a legjobbak képesek, akik tanulmányaik mellett jellemzően 1 tárgyban (heti 2 órában) tudnak csoportok tanításában segédkezni. Azt is figyelembe véve a hallgatók terhelhetőségét, a felsőbb éves hallgatók körülbelül 15%-a alkalmazható lehet ilyen módon, körülbelül 1200 órát el tudnak látni, amivel az oktatók heti óraszáma 12-13 órára csökkenthető.

A közoktatásban heti 18 órája van egy mesterpedagógusnak, ha mellette szaktanácsadói tevékenységet folytat. Ezzel együtt jár, hogy az egyik tanítási napon nincs órája. Heti 19 órája van a kutatótanárnak, aki tanítás mellett kutat, szaktárgyi, pedagógiai témában publikációi jelennek meg. Heti 18 óra volt 2012 előtt a középiskolai tanári óraszám. Az életpályamodell bevezetése előtt heti 12 óra volt a vezetőtanárok heti óraszáma; ők azok, akiknek az óráján egyetemista gyakorolja a tanítást, így vele előre egyeztet, utána megbeszélik a történteket. Ez alapján napjaink egyetemi oktatója nem egy olyan kutató, aki kutatásai mellett tanít is, hanem egy olyan tanár, aki kutatásokban is részt vesz. Ilyen mennyiségű oktatási tevékenység mellett a kutatás-fejlesztési tevékenységnek egyenlően kellene megoszlania a szakma és a pedagógia között, az egyetemi oktatóknak pedagógusoknak is kellene lenniük, azzal a különbséggel, hogy a kisgyermekek és kamaszok nevelésének hagyományos pedagógiája helyett az andragógiában [153] kell jártasságot szerezniük.

Modern pedagógia a felsőoktatásban

A lényeges szempontokhoz a tudományos cikkekkel [153] tartalmában azonos, de a jelentősen tömörebb megfogalmazást találunk a Wikipédián:

1. *A felnőtt, a gyerekkel szemben, önálló, önirányító személyiség, de ha gyermekként kezelik, úgy is viselkedik. A felnőttek önirányítási képességéből adódik, hogy a tanulás sikeressége miatti felelősség nem csak a tanárt terheli, hanem a tanulót is. Építeni kell az önálló tanulásra a megfelelő segítség és irányítás mellett. Ehhez szükséges módszer a párbeszéd, a visszacsatolás és a személyre szabott értékelés.* ***Feladat:*** *Az önirányítás képességének kifejlesztése. (Míg a pedagógiában a tanuló függő személyiség. A tanár dönti el, hogy mit, mikor és hogyan tanuljon és az értékelést is ő végzi. Jellemző módszer az előadás, bemutatás.)*

2. *A felnőtt tanulók rendelkeznek az életről, a szakmáról számos tapasztalattal, ami hasznos tanulási forrás lehet, ugyanakkor meg is nehezítheti az új tudástartalom befogadását. A felnőttek tapasztalatainak létéből következik, hogy a felnőttképzésben az előadói módszerek helyett a párbeszéd, illetve a csoportos módszerek kiemelt fontosságúak, hiszen ily módon a csoporttagok egymástól is tanulhatnak.* ***Feladat:*** *Nyílttá kell tenni a tapasztalatokat. (Míg a gyerekek kevés tapasztalattal rendelkeznek a világról, így azokra kevésbé lehet építeni.)*
3. *A felnőttek tanulása szükséglet-központú, hiszen a valamilyen meglévő probléma megoldására törekszik, az új ismeret elsajátítása számára az életgyakorlat segítője. Szükséges, hogy a tananyag is élet- illetve feladatközpontú és problémaorientált legyen.* ***Feladat:*** *A tudásszükséglet fejlesztése. (A pedagógiában inkább a tárgyközpontúság a jellemző és a tantárgyak logikája a vezető motívum.)*
4. *A felnőtteket belső tényező motiválja: önbecsülés, önmegvalósítás, meg akarnak felelni az új kihívásoknak. (A gyerekek inkább külső nyomásra, motivációra tanulnak.)*

A már korábban leírt *Haladási napló* és *Ellenőrző feladat* módszertani fejlesztésünk is példa ezekre az elvekre; egyben ez magyarázat arra is, hogy kamaszkorban és ezt követően miért jellemző a ragaszkodás a korábbi gondolkodásmódhoz!

Az egyetemistákról azt tartják, hogy a gyerekkor és a felnőttkor határán vannak, idősebb oktatók szerint egyre tovább tart a gyerekkor. Azonban pont a felsőoktatás az, ami gyereknek kezeli az egyetemistákat, eltartotti státuszban tartja, képzési feltételeivel nem, vagy csak nagyon nehezen egyeztethető a családalapítás, családfenntartás, önálló élet- és háztartásvezetés. Az egyetemi környezet a személyes fejlődésre csak kis mértékben hat. A „gyerekek” felnőtt öntudattal, életcéllal rendelkeznek; biológiai és kognitív fejlettségük alapján felnőttek.

Az andragógia ismeretét azért tartom szükségesnek, mert ez emeli ki azokat az ismereteket a pedagógiából, ami a korosztályra vonatkozóan szükségesek. De az elmúlt évtizedekben a közoktatásban is jelentős elmozdulás történt a gyerek belső motivációja, önmegvalósító fejlesztése felé. A poroszos „szoktatás” mellett egyre nagyobb hangsúly van a tapasztalatszerzésen, az ismeretek önálló megszerzésén. A modern pedagógia minden jellemző törekvése az élethosszig tartó tanulást készíti elő, a legfontosabb tudás az önálló tanulás tudása. Erről szól a munka

szervezés szempontjából a projektmunka, a „flipped classroom”, a munkaforma szempontjából a csoportmunka, párban tanulás, egyéni tanulás.

A régi, poroszos oktatást szimbolikusan is jellemzi a katonaság, aminek a nevelési célja elsősorban a „kuss és tedd, amit mondanak”, utasítások gondolkodás nélküli elfogadása volt, csak kiegészítő hatásként jelent meg az „anya szoknyájától elszakadás”. A katonai nevelésben központi szerepe van annak, hogy a külső nyomás elnyomja a belső motivációkat. A közoktatásban zajló módszertani változás hatását felerősíti, hogy megszűnt az egyéves sorkatonai szolgálat is. A mai hallgatók nem olyanok, mint a tanáraik voltak fiatalkorukban.

Az oktatókkal beszélgetve az ELTE-n és a BME-n, azt tapasztaltam, hogy az oktatói tevékenység kötelező, de szakmailag nem elismert. Az ELTE-n a tanárképzés miatt jellemzőbb, hogy az oktatók oktatásmódszertannal foglalkozó közösségekben is megjelennek, publikálnak, de a BME-n ez a tevékenység sokkal kisebb fontosságú a szakmai publikációkhoz képest. Az egyetemnek és az egyetemi tudományos munkatársaknak, a tanszékek számára a szakmai együttműködések, kutatási projektek hozzák a pénzt, az oktatás deficites. Az utóbbi években az informatikushiány drasztikus növekedésével a képzésbe betársul az ipar is – duális képzéssel próbálnak segíteni a minél gyorsabban, minél nagyobb számú munkaerő előállításában. Sajnos, sokszor az az érzésem, hogy vak vezet világtalant. A duális képzés akkor lenne hatásos, ha legalább az egyik fél pontosan tisztában lenne a felnőttképzés módszertanával.

Az egyetemi oktatók hagyományosan az egyéni kapcsolatokon, mester-tanítvány személyes tiszteleten alapuló oktatást ismerik. A tanítvány ilyen esetben szinte gyermeke, saját neveltje a tanárnak. A tömegképzéssel ez a fajta viszony esetlegessé és elenyésző arányúvá válik. Képtelenség a heti legalább 10 órában, félévente 100–600 hallgatóval ilyen személyes viszonyba kerülni. Ebben a helyzetben a nem profin felkészített oktató a korábbi tapasztalatai alapján próbál úrrá lenni a problémákon, ehhez mintát a saját gyerekkori tapasztalata adja. Nem az egyenrangú, érdekek által vezérelt felnőttet látja a hallgatókban, hanem a bulizós, játszó, lusta, szemtelen gyereket. A megoldása nem a belső motivációkra épít, hanem külső fegyelmező eszközöket alkalmaz. De ha a felnőttel úgy bánunk, mintha gyerek lenne, akkor úgy is fog viselkedni. A duális képzés a problémát azzal próbálja megoldani, hogy a hallgatókhoz „egyéni oktatót” a képző cég szakemberét rendeli, mester-tanítvány kapcsolatot kialakítva közöttük. Kérdés, hogy a tanítvány előtte fel lett-e készítve az önálló tanulásra, problémamegoldásra vagy megfelel-e a mester (sokszor ki nem mondott) elvárásainak vagy a mester rendelkezik-e tanítási készséggel, pedagógiai érzékkel.

7.3.3

XI. Az oktatási módszerek hatékonysága

XI/1. Esettanulmányok[71]

Az egyik oktatónak hiányzik a katonaság, mert ott a hallgató megtanulta, hogy a külső szabályokat a belső motiváció elé kell helyeznie. A katonaságnál fegyelmet, bírálat – és gondolkodás – nélküli engedelmességet vártak el tőle, ezt várja az oktató is az előadáson. Ezzel szemben a mai fiatalok fegyelmezetlenek, mással foglalkoznak. Valójában már az előadás elején feladják a gondolatsor követését, mert nem látják, hogyan kapcsolódik a céljaikhoz (egy külső kényszeren, az elégséges elérésén kívül), inkább mással, fontosabb dolgaikkal foglalkoznak vagy egyszerűen pihennek (alszanak vagy játszanak). Az oktató kikéri magának, hogy a hallgatók nem csak rá figyelnek, a hallgatók számára bántó, hogy pazarolják az idejüket.

Egy másik oktató az előadáson a fegyelmet „osztály vigyázz!”-zal gondolta megteremteni. Alaphelyzet, hogy az oktató az egyedüli „adó”, a hallgató egyetlen funkciója, hogy ő a „vevő”. A tanórán a beszélgetés, kooperáció, önálló vélemények kifejtése nem szokás. A hallgatók az oktatói kérdéseket, feleltetésnek tekintik. Egy hallgató megsértődött, amikor egy gyakorlatot helyettesítettem és megkértem, hogy megoldását mutassa be a többieknek is. Szerencsére a többiek vették a lapot, hajlandók voltak megosztani gondolataikat a többiekkel, de ők is jelezték, hogy ez a módszer teljesen szokatlan számukra.

Számos oktató meggyőződése, hogy előadásának meghallgatása a tudás alapja. Uralkodó elképzelés, hogy minden ismeret tanulása úgy kezdődik, hogy a tanár elmondja az új anyagot… Közhiedelem, hogy a hallgató csak akkor készül a gyakorlatra, ha beugrót kell írnia. A hallgató csak akkor ír házi feladatot, ha azt a tanár ellenőrzi. Elvárás, hogy a hallgató – csak úgy, mint a gyerek a hallott szavakat – naprakészen vissza tudjon mondani mindent, amit valaha hallott vagy olvasott.

7.3.4

XI/2. Előadás

Az egyetemeken a nagylétszámú közönségnek tartott előadás a tömegképzéssel vált szokássá, az oktatói erőforrás pótlása céljából. Az, hogy már kialakulásakor sem tartották optimálisnak, látszik az egyetemek történetéből is. Az előadások legalább 45 percesek, de rendszeres a 90 perces előadás is.

[71] Ebben a fejezetben szándékosan név nélkül írom le az eseteket.

Az előadás tartásának a lehetősége az oktató számára szakmai elismerés, mert alkalmat ad szakmai tudásának nagyléptékű, mélyebb összefüggéseket is feltáró átadására. Az előadó számára pozitív érzés, hogy ismereteit így, összefüggéseiben elmondhatja, adhatja elő; érzése szerint: átadhatja.

A hallgató számára az előadás… Ha nem kötelező és egyéb személyes (pofára értékelés) következménye sincs, akkor a részvétel körülbelül mutatja, hogy a hallgatók mennyire tartják hasznosnak. Ez nem azonos a „jó”-val, mert nem az élvezet, hanem a befogadható tudás és ráfordított idő hányadosa a mérvadó, amit befolyásol, hogy a hallgatónak milyen egyéb feladatai, elfoglaltságai (ZH-ra tanulás, ügyintézés, határidős munkák) vannak. Ha az előadás kötelező, vagy „nagyon ajánlott”, akkor a hallgatói részvétel magas lesz, de a hallgatók igyekeznek csendben hasznosítani az időt. Például írják a leckét a következő gyakorlatra, beadandóra; levelezneznek, chatelnek; igyekeznek mindent leírni, miközben statisztikát készítenek az előadó kifejezéseiből (igyekeznek ébren maradni); kártyáznak, játszanak; színeznek, rajzolnak, kötnek (konkrétan ezt is láttam).

Az ELTE-n lehetőségem volt arra, hogy gyakorlatokon, még a beugró megíratása előtt megkérdezzem a hallgatókat, hogy mi volt az előadáson. Ez a – 90 perces – előadás lehetett 2 nappal korábban, de volt, hogy csak 2 órával (egy ebédszünettel) előzte meg a gyakorlatot. A hallgatók már számítottak a kérdésre, jártak rendszeresen az előadásokra, volt, aki jegyzetelt is. Az első, amit mondtak, az a végszó volt. „Ott hagytuk abba, hogy…” Azután a „Mi volt még?” kérdésre kisebb felvezetéssel elmondták, hogy mi volt az előadás elején: Miután elmondta az előadó az adminisztratív tudnivalókat, arról volt szó, hogy… Az egyik hallgató faggatásom hatására kifakadt: „Tanárnő, ne haragudjon! Az elejére emlékszünk, hogy hogyan kezdődött, meg a végére, mert az volt legutoljára. De a közepe egy massza. Emlékszem, hogy értettem, de semmi sem jut belőle eszembe.”

Az oktató azt hiszi, hogy átadja a tudását, de valójában csak adja (esetleg leadja). Elvárja, hogy a vétel oldalán aktív, szorgalmas vevők legyenek, akik szívják magukba a leadott anyagot. Mintha lenne egy olyan elképzelés, hogy a hallgatóság már „felnőtt, ezért tovább bírja”. Valójában nem bírja tovább, sőt… még addig sem bírja, mint a gyerek, mert ha figyel, akkor azonnal több ismerethez kapcsolja az új ismeretet.

Kis kitérő: a szerepekről

A kutatásom során több oktatóval beszélgettem, próbáltam elmondani tapasztalataimat, kutatási megállapításaimat. Próbáltam összefüggéseiben elmondani, amit gondolok. Nagyon csekély sikerrel. Egy idő után a beszélgetésekhez vázlatot írtam, sőt, volt, hogy leírtam előre, hogy mit

szeretnék mondani. Mert soha nem tudtam végig mondani egy nagyobb ívű gondolatot. Az oktatók már az első mondatok után az elhangzottakat kapcsolták saját aktuális tapasztalatukhoz, visszakérdeztek, konkrét megoldásokat kértek akkor, amikor általános jelenségre szerettem volna felhívni a figyelmet, de legalábbis elmondták, hogy az adott témában ők mit tettek. Eközben azt éreztem, hogy egy dolog a fontos számukra, hogy azonnal hasznosítható ismeretet adjak, minél rövidebb időn belül. Ezért azután a beszélgetéseim az oktatókkal néha interjúk voltak, amin az oktatók arról meséltek, amit ők fontosnak gondoltak és én csak akkor kérdeztem vissza, ha nem értettem valamit. Ez olyan, mint az előadás. De többször inkább beszélgettünk, vitatkoztunk, ahol a kérdések rendszeresen eltértek a témától, az aktuális élethelyzetre irányultak és az oktatók aktuális problémáit vetítették rá az egyébként elvi kérdésekre – mennyi pénzért, milyen személyi feltételekkel… Nem gondolom, hogy ez a fajta beszélgetés rossz. Sőt, amennyiben sikerül a saját szempontjaimat is érvényesíteni, akkor hasznos, mert a beszélgetés közben megismerem a másik fél gondolatait.

A kutatás során a gyerek-felnőtt létet nagyon furcsán éltem meg. Egyrészt, 30 éve pályán levő pedagógusként ismerem az oktatói, előadói szerepet, tudom, hogyan kell a gyerekkel bánni, hogyan kell rászoktatni bizonyos dolgokra. Másrészt doktoranduszként hallgató voltam, ráadásul együtt tanultam az elsőéves hallgatókkal. Nagyon nehezen tűrtem a hallgatókra (így rám vagy társaimra) kényszerített „gyerek” szerepeket, az ilyen jelenségekre élesen reagáltam. Emiatt nagyon sok vitám volt, nagyon sokan haragudtak rám. Másrészt e kettős szerepben megértettem, megéltem, hogy a legjobb előadásom is egy idő után „falra hányt borsó”. Szó szerint: én mondom, szórt üzenetben küldöm a gondolataimat, a hallgató hallja, de a tudatáról, mint a falról a nyers borsó visszapattan.

Előadás gyerekeknek, felnőtteknek

Egy gyerek nem tud sokáig egy dologra koncentrálni, mert a környezetből érkező impulzusok elvonják a figyelmét, a korosztálynak megfelelő mozgásigényt nem tudja visszafogni. A hároméves gyerek körülbelül 5 percig foglalkozik egy témával, az első osztályosoknál körülbelül 30 perc, amíg nyugton tud ülni. A középiskolában megjelenik a tanári előadás, de ez a tanórának csak egy része. A módszeresen felépített tanóra adminisztrációból, korábban tanultak ismétléséből, feleltetésből, gyakorlásból és végül az új tananyag ismertetéséből áll. Így legfeljebb 20 perc jut az előadásra. Természetesen, van, hogy ennél hosszabb egy előadás. Van, hogy a tanár 45 percen keresztül „nyomja”, de ezeken az órákon már megfigyelhető, hogy a diákok „alszanak”. A mással foglalkozás több formáját elég jól vissza tudja fogni a házirend és az, hogy egy osztályban legfeljebb 35 tanuló van, akiknek a fegyelmezése közvetlen.

Felnőttek sokféle helyzetben hallgatnak előadásokat. A konferenciákon jellemzően 15-20 perces előadásokat szerveznek, a TEDx előadások maximális ideje 18 perc[72]. A vallási szertartásokon a tanítás rész még ennél is rövidebb. Az értekezleteken elhangzó több, mint egyórás vezetői tájékoztatókat vállalati környezetben az időpocsékolás kategóriába sorolják. Közoktatásban az ilyen értekezletekre visznek a kollégák egy-két osztályra való javítandó dolgozatot magukkal, hogy hasznosan teljen az idő.

Konklúzió

A 45 perces egyetemi előadás gyereknek (akinek kötelező figyelnie) kínzás, felnőttnek, hallgatónak időpocsékolás. Egy előadás nem lehet hatékony, ha a hossza 20 percnél több.

Az előadás fontos szerepet tölt be az ideák, tudások nagy csoportnak történő átadásában. Az előadó személyes kompetenciája, metakommunikációja biztosítja az ismeret hitelességét, ezért az előadás nem egyenértékű a róla készült videófelvétellel, az előadó nem helyettesíthető mesterséges intelligenciával.

A tudás átadása pusztán kiscsoportos környezetben napjainkban nem hatékony, a tömegképzés ezt nem teszi lehetővé. A nagylétszámú előadások megszervezése komoly koordinációt és logisztikát igényel. Az előadás időbeli hatékonyságában nem csak az előadás idejét kell figyelembe venni, hanem az egyéb tevékenységből kieső időt, például a közlekedéssel töltött időt is. Ezért az előadás hatékonysága elvileg nő, ha hosszabb az előadói idő.

Az egymásnak ellentmondó peremfeltételek feloldását jelenti, ha – konferenciaszerűen – egymás után több kis blokkban tartjuk az előadást. Gondoskodunk arról, hogy a blokkok között a hallgatóság kikapcsoljon a „vétel” módból és a hallottakat feldolgozza. Ez nem a „Van kérdés?... Oké, mehetünk tovább” fejezetek közötti váltás, hanem konkrét, hallgatók produktív aktivitást igénylő feladat adása a hallgatóságnak.

A „Van kérdés?” előadói kilépés az „adás” állapotból csak azoknak a hallgatóknak segítség, akik vétel közben listázni tudják a kérdéseket, felvetődő problémákat. A többi hallgató számára ez egy „két adós”, dialógus jellegű része az előadásnak.

Az előadás blokkosítása, aktivitással megtörésére a látogatott és megfigyelt előadásokon többféle (jó, elmegy, rossz) megoldást láttam.

- A *Programozás alapjai 1.* előadáson az éppen elhangzottakkal kapcsolatosan gondolkodtató kérdések, példák vannak beépítve, amelyeknél szavazni lehet a helyes válaszra és az

[72] https://www.ted.com/participate/organize-a-local-tedx-event/before-you-start/tedx-rules [2021.10.30]

egyes válaszokat az előadó minősíti. (Például: azért örülök, hogy legtöbben… választottátok…) Az előadás élvezeti értékét ezek a kis közjátékok jelentősen növelik. A párhuzamosan tartott előadáson a másik előadó ugyanezeket a kérdéseket felteszi, de a válaszokat nem kíséri értékelés, csak a lehetséges válaszok elemzése. Ezért a hallgatók inkább költői kérdésnek tekintik.

- A Rendszermodellezés előadáson az előadó szintén tett fel kérdéseket előadása közben, ezzel aktivizálva a hallgatókat. Ezek a kérdések vagy egy régebben tanult ismeret felelevenítését szolgálták, vagy az éppen soron következő tananyag felvezetéseként („mit gondolsz…?") az aktív figyelem visszacsatolását célozták meg. Az ilyen típusú kérdések nagyon pozitívan hatnak az előadás minőségére, a válaszok segítik az oktatót abban, hogy kapcsolatban legyen a hallgatókkal, a hallgatókat pedig abban, hogy ne aludjanak el. Azonban ezek a kérdések jellemzően nem $LAU^{L}2$ célúak, a hallgató nem tudja kipróbálni, hogy az addig hallottakat megértette-e.
- Távoktatási környezetben, egy Ruby on Rails tanfolyamon találkoztam az előadást megszakító papíros szavazással. A videó felvételen elhangzott a kérdés és a lehetséges válaszok, majd távoktatási környezetben a folytatáshoz válaszolnom kellett egy űrlapon, míg az előadáson élőben résztvevők (a következő videórészleten láthatóan) sárga-piros-zöld lapokkal szavaztak. A videón ezután 5 percben a válaszok kiértékelése, magyarázatok voltak láthatók.
- A WiPSCE/ISSEP konferencia hosszú záróelőadásán egyes blokkok után a hallgatóknak egymás között kellett megbeszélni a hallottakat, a csoportok véleményt fogalmazhattak meg.
- A Programozási alapismeretek, a Bevezetés a számelméletbe, az Analízis és a Fizika előadásokon az előadó legfeljebb költői kérdést tett fel. Az előadáson a hallgatónak a folyamatos „együtt gondolkodás", a gondolatmenet követése volt a feladata.

Az informatikai eszközök oktatásba bevonásával akár 5 percben is lehet nagy létszámú hallgatóságnak feladatot adni, a válaszokat begyűjteni és kiértékelni. A legismertebb ilyen célú szoftver a Kahoot, de egyetemi fejlesztésekben is láttam hasonlót. Fontos, hogy ezek a feladatok elsősorban az önértékelést kell szolgálják, a hallgatónak önállóan kelljen a feladatot megoldania, a megoldásról kapjon visszajelzést, de a hibás válasz ne jelentsen hátrányt, hátrányos megkülönböztetést. Az előadásnak, a feladat kiadása környezetének megfelelő kialakításával a feladatok alkalmasak lehetnek a jelenlét ellenőrzésére (természetesen, aki helyeben alszik, az nincs jelen), a válaszok tárolása alkalmas lehet lemaradások felderítésére, egyénre szabott feladatok kiadására.

A távoktatás, illetve előre felvett előadások készítése során még inkább oda kell figyelni a hallgató figyelmének fenntartására, aktivizálására. Itt még fontosabb a rövid blokkok készítése, a köztes önálló kipróbálás biztosítása. Mivel ebben az esetben az oktatónak nincs közvetlen kapcsolata (vagy nagyon csekély) a hallgatóval, a tanulást motiváló eszközöket tudatosan kell alkalmazni. Az online távoktatásban a feladatokra érkezett válaszokat valós időben érdemes értékelni, a felvételről lejátszott előadásokhoz hasznos lehet a gamifikációs elemek beépítése.

Az előadás szerepe a LAU-modellben

Az előadás a tanulás folyamatának jellemzően a kezdőlépése (LAU^L1). A kiadott feladatok célja a megértés visszaellenőrzése, kipróbálása, LAU^L2. De a tudás megszerzése ma már nem csak, pontosabban főleg nem az előadáson történik. Az egyszer, az előadáson hallott tananyag szinte csak az érdeklődés felkeltésére jó. Mindegyik informatika előadásra jellemző, hogy gondolkodási folyamatot kell megérteni, később a teljes folyamatot tudni kell reprodukálni. A nagyelőadós előadás tipikus problémája, hogy nem lehet mindenkinek megfelelni színvonalban, beszédtempóban, stílusban. Az egyik hallgató már hallott valamit a témából, szeretne gyorsabban haladni; a másik elakadt egy számára új fogalmon, neki jobb lenne egy pillanatra megállni, egy kis kitérőt tenni az adott fogalomra.

Egy előadás „emészthetőségén" sokat javítanak a példák, történetek. Ezek a haladóbbaknak is érdekesek, megerősítést ad (LAU^L1A); akik pedig újként hallják a témát, azoknak a példa – ha korábbi tapasztalataikhoz kapcsolódik –, segít az összefüggések átlátásában (LAU^L1M). Ilyen történetek például, a regisztrációs hét mizériája a Neptunban; az előadó „találkozása" a kávéfőzőgéppel. Azonban láttam olyan előadást is, ahol a történetek, példák elnyomták a lényeget.

Példa anekdotikus előadásra:

> Diák kiselőadást tart a processzorok működéséről. Az előadásában számos márkanév, árak és marketingjellemző elhangzik, esetleg vásárlásra vonatkozó tanácsot is ad. Arról azonban egy szó sem hangzik el, hogy a gép működésében milyen szerepe van a processzorok említett tulajdonságainak. A többmagos processzor gyorsabb, menőbb, de nem derül ki, hogy mennyiben működik másképp, mitől gyorsabb… Az előadásból csak néhány percnyi hasznosítható a dolgozatban: a processzornak van sebessége, minél gyorsabb annál jobb. Ha nem diákelőadásról van szó, akkor az ilyen típusú előadáson sokszor elhangzik, hogy „ezt dolgozatra nem kell megjegyezni". Diákelőadás esetén a tanárnak utólag tisztáznia kell, hogy az előadásból mi volt a számonkérhető tartalom.

Az elmesélt példáknak azon kívül, hogy elnyomhatja a lényeget, van egy másik veszélye is. Rengeteg olyan példa van, ami az előadó számára érdekes, személyes élményen alapul, de a hallgatóknak semmitmondó. Például, amíg nincs saját nevelésű gyereke valakinek, addig nem érzi át annak az élményét, hogy a gyerekben kialakul a számok mennyiségi fogalma. A kávéfőzőgép működése sem érdekes annak, aki csak kotyogós kávéfőzőt használ vagy azt se. Ezért a példák a hallgató korábbi tapasztalatától függően LAU^L1A, LAU^L1M vagy, ha teljesen idegen számára a környezet, akkor LAU^L1P tanulást eredményez.

A tanulás nagyobb része a tanultak alkalmazásakor, az előadáson hallottak visszaidézésekor történik. Ehhez nagyon hasznos, ha van az előadás tartalmával azonos részletességű jegyzet, vagy elérhető videón (részletekben, téma szerint címkézve) az előadás. Minden esetben jól láthatónak kell lennie annak, hogy melyek a lényeges gondolatok, melyek a példák; mit kell tudni és mi az, ami csak a megértés segítésére szerepel.

Az ELTE IK-n körülbelül 1/3, a BME VIK-en 1/2 a kontaktórákban az előadás aránya. Azt gondolom, hogy mindkettő nagyon sok. Helyette több mentorált, önálló ismeretszerzés kellene, ahol a tanár a hallgató kérdésére pont azt mondja, adja elő, amire a hallgatónak szüksége van.

Nem szabad megfeledkezni arról, hogy az előadások szervezése elsődlegesen az oktatói kapacitás szűkössége miatt fontos. Ezért nem képzelhető el, hogy minden hallgatónak „személyes előadója” legyen. Azonban nem szükségszerű, hogy az előadás az ismeretek felvezetése, tanulási folyamat vezérlője legyen. A „flipped classroom” éppen a folyamatvezérlés megfordításáról szól: A tanuló önállóan olvasgat, tájékozódik egy adott témában, megismerkedik problémákkal, feladatokkal. A kontaktórán feladatokat old meg, amiben segíti a tanára, mentora. A kontaktóra része, hogy a hallgatók kérdezhetnek a már tanulmányozott témával kapcsolatban, a kérdésekre a tanár válaszol.

Mi lenne ha…?

Mi lenne, ha az előadások előtt a gyakorlatokon, önálló – esetleg projekt – tanulás során találkozna a hallgató az előadáson tárgyalt témákkal? Mi lenne, ha az előadások előtt megjelentetnék a tartalmi összefoglalóját? Ahogy a konferenciákon is előre el lehet olvasni, hogy miről fog szólni az előadás, ahogy a cikkek elején megjelenik az absztakt. Egy-egy előadás, mint a konferencia szekciók 2–4 ilyen témából állna.

Mi lenne, ha az előadáshoz előre megjelenne néhány tanulmányozható példa, eset? Mi lenne, ha a hallgatók előre tudnák, hogy az adott előadáshoz milyen előismeretek szükségesek, konkrétan az adott előadás mire épít?

Mi lenne, ha az előadáson nem az oktatók adnák le, nyomnák a hallgatókba az ismereteket, hanem a hallgatók szívnák magukba? Ha a hallgatók kimondottan az adott téma iránti érdeklődés miatt ülnének be az előadásra, néznék meg a felvételt.

7.3.4

XI/3. Gyakorlat és labor

Beugró

Mindkét egyetemen, az összes gyakorlaton, amit látogattam vagy tartottam, elvárás, szokás beugrót íratni. A beugró célja elvileg, hogy ellenőrizze, hogy a hallgató az előadáson elhangzottakat megtanulta-e. Másképp, a beugró célja, hogy ellenőrizze, hogy a hallgató tudása alkalmas-e a gyakorlat feladatainak végzésére. Ez a két megfogalmazás nem teljesen azonos, de a lényeg, hogy a beugrók eredménye valamilyen módon beleszámít a félév értékelésébe. Ezzel rákényszeríti a hallgatót a gyakorlatra történő előzetes otthoni (vagy óra előtti) készülésre. Tipikusan külső kényszer a tanulásra: a beugró a lusta gyerekeknek szóló fenyegetés.

A beugró elvileg a házi feladat megoldását méri. Elvileg az előadáson hallottak megtanulása ugyanis házi feladat. Mint már korábban írtam és a LAU-modellből is látszik, az előadáson hallottak megtanulása a gyakorlatok során történik. Mit kell erre otthon tanulni? Jellemző, hogy az előadás jegyzetét kell átnézni, megtanulni. Az előadás jegyzete hiányos, ugyanakkor a kiegészítéseket is tartalmazza. Be kell magolni a megírt jegyzet több tíz oldalát, be kell magolni a prezentáció diasorát, mind a 40–60 diát? Emlékezni kellene az előadáson elhangzottakra?

Vajon mit gondol fontosnak egy előadásból az oktató és mit jegyez meg a hallgató? Mennyi időt kell tanulással tölteni, hogy a beugrót sikeresen megírja a hallgató… Lehetne folytatni a kérdéseket, amelyek abból adódnak, hogy a hallgatónak egy olyan házi feladatot kell megcsinálnia, amelyet nem specifikált pontosan az oktató. Nagyon tágan értelmezhető az, hogy a tanuld meg azt, „ami az előadáson volt”. Ami még rosszabb, hogy alapvetően magolást jelent az ilyen típusú feladat.

Óraadóként nem igazán tudtam mit kezdeni a beugró feladattal, hasznát nem láttam, kárát annál inkább. Ezért azután renitens is voltam ebben a témában, de azt gondolom, hogy nem csak én. Mindkét egyetemen vannak a beugró íratására és értékelésére szabályok és van a „józan ész”, ami szerint figyeljünk, hogy nem bukjon meg csak a beugró miatt egy hallgató.

Az ELTE-n könnyebb a beugrót „átértelmezni”, mert a gyakorlatvezető tervezi a gyakorlatot, a beugró kérdését is ő találja ki. A beugró feladatának az előadáshoz kell kapcsolódnia,

ezért specifikációval vagy struktogrammal kapcsolatos rövid, 5 perc alatt megválaszolható kérdés kell. Ahol úgy éreztem, hogy rendszerező, áttekintő felkészülés szükséges, ott kiadtam házi feladatnak a rendszerezést. Például „szótár készítése" a bemeneti adatok, változók jelöléseiből. Vagy: az előadásjegyzetből minden tétel legtömörebb specifikációjának és a struktogramjának az kimásolása. A beugró ezeknek egy részét kérte számon, de aki hozta a jegyzetet, annak nem kellett beugrót írnia vagy használhatta a jegyzetét.

Házi feladat

A házi feladattal kapcsolatban általában az az oktatók ellenérzése, hogy azt a hallgatók egymásról másolják, egymását adják be. Abban az esetben, ha a házi feladat nem hasznos a hallgatónak, akkor a másolás esélye elég nagy lesz. Egy esszé-kérdés jellegű házi feladat útmutatást ad a hallgatónak arra, hogy hogyan tanuljon, mire figyeljen, mit emeljen ki az előadás anyagából. Ezért a feladat megoldását a hallgató is értelmesnek fogja érezni. Ugyanakkor egy esszé, egy kidolgozandó prezentáció egyedi, a másolatot több módon is fel lehet ismerni: túl nagy egyezés a megjelenésben, metaadatok (például mentési idő) azonossága, a készítő tájékozatlansága „saját" munkájában. A tanulásnak ebben a fázisában azonban nem baj, ha együtt, egymás munkáját is felhasználva készítik a hallgatók házi feladatot. Az a probléma, ha a másolás során az átmásolt adat például egy vágólapon keresztül kikerüli a másoló agyát. A tanulást segíthetjük azzal, ha ugyanazt a feladatot többféle formában kérjük. Például egy struktogramot rajzolva és lefényképezve, vektorgrafikusan, pixelgrafikusan, valamint speciális szoftverrel elkészítve is. Ugyanannak az ismeretnek sokféle használatát is fel lehet adni házi feladatként: tesztkérdések formájában, hibakeresési feladatként, konkrét gyakorló feladatként.

A házi feladat ellenőrzése a hallgató érdeke kell, hogy legyen. Ezért nem az a fontos, hogy hibátlan legyen, hanem az, hogy kiderüljön, ha hibás, a hibákat meg lehessen beszélni, ki lehessen javítani. Az ellenőrzésre nagyon jól használható lehet egy kiértékelő szoftver, de az eredmény önmagában nem minősít. A megoldások ellenőrzése lehetséges csoportmunkában, egy-egy részlet „beugró" jellegű számonkérésével.

A BME VIK *Programozás alapjai 2* tárgy első laborgyakorlataihoz[73] volt házi feladat, amelyet meg lehetett oldani, majd meg kellett oldani a beugró teljesítéséhez. Későbbi órákon a feladatok a laborgyakorlat anyagát jelentették. A feladatok alkalmasak otthoni munkára. Több részfeladat valójában nem feladatmegoldás, hanem egy-egy jelenség kipróbálását, a tapasztalatok leírását várja el. A feladatok önálló feldolgozása mellett természetesen adódnak problémák,

[73] Tantárgyfelelős: Dr. Szeberényi Imre; a jporta.iit.bme.hu portált fejlesztette: Estók Dániel

amiket a laborgyakorlaton kell megbeszélni. Ha a feladatok végig házi feladatként lennének kiadva, akkor a „beugró” a feladat kötelező része a kipróbálásra írt feladatok és azt követően a feladatok megoldása vagy a feladattal kapcsolatos kérdések írása lehetne.

Az ELTE IK Programozás tárgyához is egyre több (most már négy) házi feladat tartozik, amelyek megoldására 2-2 hét áll rendelkezésre. Ezek a feladatok beszámítanak az értékelésbe, ezért – a másolás elkerülése érdekében – nagyon sok feladatból véletlenszerűen kapnak 1-1 megoldandót a hallgatók. Az értékelés miatt a megoldás konkrét módja, minősége háttérbe szorul, csak az számít, hogy a kiértékelő szoftver teszteredménye milyen. Mivel a csoport minden tagja más feladatot kap, ezért a megoldások megbeszélése, elemzése csak egyedi konzultáció formájában történik, de inkább nem történik sehogy sem. Mivel a feladat önálló megoldásra van kiadva, a hallgatók nem publikusan kérnek segítséget, ha problémába ütköznek. Ez a segítség jöhet ismerőstől, BME-s szobatárstól, aki olyan megoldást ajánl, ami működik, de esetleg nem fogadható el (például kivételkezelés, ciklus megszakítás, lambdakifejezések használata). Vagy jöhet csoporttárstól, évfolyamtárstól ötlet, ami az előzők miatt nem ellenőrzött. Ilyen formában minden évben volt valami fertőzés módra terjedő megoldási mód, amit a gépi tesztelés ugyan elfogadott, de a beadandó feladat kódjaként nem lehetett értékelni. Ha a házi feladat nem kisdolgozat jellegű lenne, akkor a hallgatók az oktatók számára is látható módon beszélhetnék meg a megoldási módokat, jobban lehetne támogatni a helyes gondolkodásmódot.

Kiugró

A „Kiugró” a „Beugró” mintájára alkotott mérési pont, amelyet annak jelölésére használtunk, hogy a laboron, gyakorlaton való részvétel érvényességét nem az óra kezdetén kellene validálni, hanem az óra végén. A beugróval nem csak az a probléma, hogy a hallgató nem tudja, hogy mire készüljön és nem is csak az, hogy a teljesítése nem jelent számára tudásbéli hasznot, hanem az is, hogy az oktató sem tudja, mit kezdjen a hallgatói munkák eredményével. Az ELTE IK-n a beugró eredményét csak a következő alkalomra kell megmondani, a BME VIK-en a labor alatt, minél hamarabb illik (kell) tájékoztatni a hallgatót, hogy sikerült-e a beugrása. Az eltérő értékelési határidő mögött az áll, hogy az ELTE IK-n a beugrók 50%-át kell teljesíteni az aláíráshoz míg a BME VIK-en a sikertelen beugró érvénytelen részvételnek felel meg. Ez a szemlélet körülbelül a testnevelési felszerelését otthonfelejtő diák esetét idézi, csak a felszerelést kell a tudásra lecserélni. A közoktatásban érzékeny kérdés, hogy lehet-e igazolatlan órát adni a felszerelést otthonfelejtő diáknak. Nem lehet. Sőt, ha a testnevelés órát lehet a folyosón is tartani, akkor utcai ruhában is lehet mozogni (például lépcsőn fel-le járni).

Az egyetemi környezetben valószínűleg a műhelygyakorlatok rendje alapján alkalmazzák a szabályt. Műhelygyakorlatok esetén a beugró felmérésen ellenőrzik, hogy a hallgató ismeri-e az eszközök működését, használatának módját, a balesetvédelmi előírásokat és rendelkezik-e megfelelő öltözékkel, saját eszközökkel a feladatok biztonságos végrehajtásához. Amennyiben ezekből bármi hiányzik, nem engedik a gyakorlati helyszínre, ténylegesen hiányzik a gyakorlatról, a feladatokat nem tudja elvégezni.

A programozás, informatikus tantárgyak jelentős részénél azonban a laborgyakorlat nem felel meg a fenti műhelygyakorlatnak. A mérnökinformatikus inkább tízujjal gépelő informatikus, mint kétkezi, fizikai munkát végző mérnök. A fentiek teljesítése lényegében a házirend ismeretét és betartását jelenti, de ehhez nem kell beugrót íratni. Az aktuális tanórai munkához szükséges előismeret hiánya valójában nem veszélyezteti sem saját, sem társai életét, sem az eszközök működését. Egyedül az oktatója idegeit tépázza az, ha láthatóan készületlenül érkezik és tudatlanságával zavarja a többieket, leköti az oktató figyelmét vagy az eszközöket nem a tanórai anyag végzésére használja (azaz játszik…).

A beugró íratásának rendszere ebből a szempontból tripla oktatói öngól. Ha a hallgató sikeresen megírta a beugrót, akkor a laborgyakorlaton jelen van. Akkor is, ha ezután semmi érdemlegeset nem csinál, ha kimegy a mellékhelyiségbe és vissza se jön. Ha a hallgatónak nem sikerült megírnia a beugrót, akkor a laborgyakorlaton nincs jelen, de ettől még ottmaradhat, sőt, javasolt is maradnia, hogy behozza a lemaradását, mivel korlátos számú hiányzást enged csak az értékelési rendszer. A harmadik káros hatás, hogy a beugrókat helyben, azonnal javítani kell és tájékoztatni kell a hallgatókat az eredményről (hogy tudják, számít-e, hogy ott vannak). A javítás közben a hallgatók elkezdenek dolgozni. Többen kérdésekkel érkeztek, másoknak az első feladattal kapcsolatban van értelmezési kérdésük… A hibátlan megoldásokról gyorsan lehet visszajelzést adni, de a részben jó és a teljes félreértésről tanúskodó megoldásokat meg kellene beszélni az adott hallgatókkal. Leginkább azokkal kellene foglalkozni, akiket el kellene küldeni.

Nekem mindig sokáig tartott a beugrókon túljutni – javítani, kiosztani, regisztrálni –, mert természetesnek tartottam, hogy a hibás megoldásokról elbeszélgessek az adott hallgatókkal. Miért úgy írta, ahogy, hogyan tanult rá, mit nem értett meg a feladatból… Ezek alapján láttam, hogy ki az, akinek olvasásra vagy kipróbálásra kell adnom feladatokat – mert nem készült –, ki az, akinek el kell magyarázni egy-egy részt, esetleg speciális feladat kell és ki az, akit egy veszsző vagy egy szinonima zavart meg. Bevallom, többször ki lehetett javítani a sikertelen beugrót részben sikeresre.

Példák beugró javítására

Az egyik első laboron egy hallgató a for-ciklusba a ciklusból kilépés feltételét írta. Mint kiderült, valahol úgy hallotta, hogy így kell megadni (talán Scratch?) és nem vette észre, hogy pont a fordítottja kell. Szándékosan, egy rossz hittel írta rosszul. Megbeszéltük, elfogadtam.

Szintén az egyik első laboron történt, hogy a hallgató nem tudta megírni a for-ciklust, ami kiírja a számokat 10-ig. A hallgató egyébként tudott programozni, nem értette a feladatot. Azt a feladatot meg tudta volna csinálni, hogy „írj program részletet, ami kiírja a számokat 10-ig”, de a „for-ciklus” kitételre úgy gondolta, hogy neki most a ciklus-fejbe kell írni a kiírást – papíron, így nem tudta ellenőrizni sem – így a kódjában minden benne volt, de a szintaktika értékelhetetlen lett. Ha egy ejnye-bejnye mellett elfogadom a megoldását, akkor nem érzi a probléma súlyát, hiszen csak félreértette, túlkomplikálta a feladatot. Ez a tolerancia azonban visszaüt a dolgozatoknál. Ha nem fogadom el, az agressz-szivitást szül, mert minden tőle telhetőt megtett, tudja a kérdésre a választ; ezért haragudni fog önmagára is és a „rendszerre” is, valójában a kettő inkompatibilitásra. Ilyenkor várható, hogy a tehetetlen harag csapkodásban, önmaga fejbeverésében tör ki. Olyan megoldást kerestem, ami fizikai módon is emlékezetes marad, de nem marad írásos nyoma. Ezért az elfogadásért 10 fekvőtámaszt kértem. Az alku később is hasznos volt, mert a fekvőtámasz pont olyan értelmetlen, mindentől független elem volt, mint a beugrók (ne keresd az értelmét, ne komplikáld, csináld). Amint komplexebb – értelmes – feladatot kellett megoldania a hallgatónak, nem volt semmi baj a tudásával, viszont a beugrókat rendszeresen elbonyolította. Én tudtam, hogy ő tudja, de ő is tudta, hogy mit nem tud.

Mindkét – és még sok más – eset mutatja, hogy egy beugró semmilyen formában nem minősíti a laborgyakorlaton való részvételt. Az, hogy értelmesen, a tananyagban haladással tölti-e az idejét a hallgató, az a beugróból nem jósolható meg ezért a részvétel minősítésére a beugró nem alkalmas, erre a gyakorlat végén van csak lehetőség. A „Kiugró” azt mutatná, hogy a labor végén elért-e egy szintet a hallgató.

A C11 és C++11 programozása tárgyban 2018-ban nem volt beugró, viszont minden gyakorlat végén be kellett adni a feladatok megoldását. Ki-ki ameddig jutott. A gyakorlat alatt az oktató figyelte a munkát, segített annak, aki kérte és szólt, ha valakinél azt látta, hogy rossz úton halad. Azt hiszem a megoldások elfogadása az órai munka alapján történt, a beadás csak a jelenlétet igazolta, de elvileg a feladatok jelentős része géppel tesztelhető. (Például a parancssori fordítás gyakorlata nem ilyen, valamint néhány feladat esetén a vélemény megfogalmazása

sem tesztelhető.) További előnye a beadásnak, hogy a hallgató később visszatöltheti, a megoldások elemzése alkalmas lehet tanulási problémák elemzésére, ZH felkészüléshez tanácsadásra… De ehhez már komoly fejlesztés, jól paraméterezhető LMS rendszer kellene.

Gyakorlat teszi a mestert

A gyakorlat és a laborgyakorlat az ismeret megtanulásának mentorált terepe. Nem csak a tananyagot kell ilyenkor megtanulni, hanem a tanulást is. Az idő kevés, ami nem fér a tanórába, azt önállóan kell megoldani. A programozás tanulásáról a legfontosabb tudnivalókat *Programozás alapjai 1.* ZH-ra felkészüléshez írt Tanácsok a tanuláshoz segédletében[74] találtam:

> *Nagy hiba a honlapon lévő anyagokat vagy más forrásokat csak olvasgatni. Nem jó ötlet a feladatok mintamegoldásait nézegetni! Nem vezet sehova, nem ad semennyi programozási tapasztalatot! Nem tanít meg gondolkozni, nem tanít meg fölfedezni, megérteni a tananyagot.*
>
> *Attól még, hogy egy megoldást láttál már leírva, nem biztos, hogy tudod reprodukálni is. Sőt akár, ha értetted is, akkor sem biztos, hogy egyből – a ZH stressze alatt – le tudod írni újra egy tök üres lapra. A programozás kreatív tevékenység, ha mindig megnézed a megoldásokat, éppen a lényegét nem gyakorlod.*
>
> *Igenis próbáld megoldani a feladatokat! Ha tudod, mi a megfelelő eszköz (pl. pointer), de nem jól használod, keresd meg magad a hibát! Az előadásanyagot nézd, ne a megoldást! Ha úgy sem megy, nézd meg a megoldás magyarázatát, ha van – ott le van írva az alapgondolat vagy a trükk, ha épp valami trükk kell. Ha úgy sem, nézd meg a megoldás releváns részét – és továbbra is, próbáld meg magad befejezni a kódot!*
>
> *Mindezt csakis gép mellett tedd. Ott a fordítóprogram, nem lehet kérdés, hogy működik-e a programod vagy nem. Próbáld ki! Használd a nyomkövetőt! Rakd tele a programjaidat printf()-ekkel, hogy lásd, mit történik a változókkal, merre megy a végrehajtás, egyáltalán mit csinál a program, amit beírtál!*
>
> *Engedélyezd a fordítód figyelmeztetéseit (lásd az előadást). Egy csomó mindenre meg fognak tanítani. Ne csak a build&run, hanem a sima build gombot is használd a fejlesztőkörnyezetben! Különben nem fogod látni a hibaüzeneteket.*

[74] https://infoc.eet.bme.hu/nzhtanacs/ [2021.10.30]

A tanácsok alapján a feladatok megoldásán kívül a gyakorlatokon (ha még nem menne) meg kell tanulni gondolkodni – és nem magolni –, felfedezni, értelmezni a leírtakat. Tudni kell egy gondolatsort reprodukálni – azaz nem a kódot kell megtanulni, hanem a kód keletkezésének okát, döntések sorozatát, a kód létrehozását. Meg kell tanulni tesztelni, hibát keresni, hibaüzenetet értelmezni, nyomkövetés különböző eszközeit használni.

A látogatott gyakorlatokon a BME VIK-en általában a kitűzött feladatok álltak a középpontban. Ha egy hallgató nem tudja önállóan megoldani a feladatot, akkor az oktató segít, felírja a táblára, kivetíti, elmagyarázza a megoldást. A gyakorlatokon lényegében ez az állandósult állapot, mert vannak olyan hallgatók, akik miatt a gyakorlatvezető úgy érzi, hogy frontálisan el kell magyaráznia a megoldást. A laborgyakorlaton azt láttam, hogy szakmailag magas színvonalon segítenek a demonstrátorok, gyakorlatvezetők. De… ezzel csak kismértékben segítették a tanulást. Azt nem tudom, hogy az órák megtartására mennyire volt hatással a jelenlétem. A hallgatókkal is beszélgettem, az ő véleményük alapján általában számított, hogy bent ül egy „megfigyelő”; fokoztam az oktatók magyarázási kedvét.

Vezetőtanárként és saját gyakorlatomból is ismerős a helyzet: egy tanuló kérdez, ezért elmagyarázom. Ha szerintem fontos kérdésről van szó vagy várhatóan több tanulónak gondot fog okozni, akkor megállítom az önálló munkát és elmondom a választ. A látogatott órák között volt olyan, ahol az első kérdésre fél perc helyett több mint öt perc volt a válasz, frontálisan – talán, hogy én is halljam. A második kérdésre is mindenkinek szóló részletes választ kaptak a tanulók. Hátul ülve láttam, hogy a harmadik kérdés válaszára már nem figyeltek a tanulók. A végén már az a tanuló sem figyelt, aki kérdezett. A kérdésre a választ megkapta, a hozzáfűzött kiegészítés nem volt releváns számára. Ugyanakkor az egyik tanuló, a válasz végére jutott el addig, hogy megkérdezze ugyanazt.

Azt láttam, hogy az egyéni munka során felmerülő kérdésekre nem hatékony frontális választ adni. Nem támogatja a kreativitást, a tesztelést és hibakeresést, az önálló tanulás képességének fejlesztését. A tapasztalatok alapján más módszereket használok:

- A gyakorlatok feladatait átnézve – demonstrátorként előre megoldva, tanárként ismerve – tudom, hogy hol várható kérdés. Van, amikor előre jelzem, hogy melyik ponton, részfeladatnál számítsanak a hallgatók az egyéni munka felfüggesztésére, máskor csak megvárom a válasszal a következő néhány kérdezőt.
- Aki nagyon elől jár a megoldásokkal, annak inkább forrást mutatok, ahol utána tud nézni (és bővítheti is a tudását).

- Aki lemaradt a válaszról, ahhoz megkérem az első kérdezők közül az egyiket, hogy ő mondja el. Ezzel az is gyakorol, tanul, aki a másiknak segít. Van, hogy meg is hallgatom, hogy hogyan magyarázza el a társának azt, amit pár perccel korábban tőlem hallott, de már ki is próbálhatott.
- Akinek egyéni problémája van – nem működik, nem úgy működik a programja, ahogy kellene –, azoknak el kell mondaniuk pontosan a problémát. Ennek célja, hogy megtanuljon jól kérdezni. A kérdésére nem a választ mondom meg, hanem a forrást. Például: előadás jegyzetben keress rá; C-puskát nézd meg, hogy mit ír; interneten keress rá, nézzük melyik oldalt érdemes nézni.
- Ha a feladat értelmezésével van gond, akkor a dilemma részletezését kérem. Háromféle hiba gyakori: a feladat bevezetőjét nem veszi figyelembe; egy-egy szó, fogalom megértése okoz gondot; a mondat tagolása nem egyértelmű. A segítség az első esetben: „Mutasd, hogy kezdődik a feladat!” A másik két esetben azt nézzük meg, hogy a tanuló értelmezése ellentmondásra vezet-e, ha kétféle értelmezés lehetséges, akkor a megoldásban ezek milyen módosítást eredményeznek. Érdemes mindkét megoldást kipróbálni, mert ez biztosítja a megfelelő (pozitív és negatív) tapasztalatot.
- A feladatokat egyénileg kell megoldani, de a feladat értelmezését meg lehet beszélni. A kész megoldásokat célszerű legalább egy társ megoldásával összehasonlítani, az eltéréseket megbeszélni.
- Kódot másolni tilos. Nem csak a copy-paste tiltott, hanem a szomszéd képernyőjéről másolás is. Egy kódrészlet megtekintése és leírása (esetleg reprodukálása) között legyen legalább annyi idő, amennyi idő alatt az adott kódrészletet el lehet mondani. (Olvasás után másfelé nézve elmondani, utána leírni.)
- Saját kódot sem szabad másolni. Ha egy gyakorlaton 50-szer kell for-ciklust írni, akkor mind az 50 esetben végig kell írni.

A programozás oktatásában és tanulásában az a nehéz, hogy egyszerre háromféle tanulási módszert kell alkalmazni.

1. Az elveket, gondolkodási módszereket hosszú távon, készség szinten kell tudni, ehhez változatos alkalmazási helyzetek szükségesek.
2. A programozási nyelv alapszavait, szintaktikai szabályait be kell magolni, sőt, az az igazi, ha nem fejből írjuk a kódot, hanem „az ujjainkban van”, ami sok gyakorlással, ismétléssel érhető el.

3. Végül, ahhoz, hogy egy programot meg tudjunk írni a programban használt változóneveket, függvényneveket a munka idejére fejben kell tartani.

A gyakorlatokon a gondolkodási módszereket legjobb szóban, személyes (humán-humán) interakciókban fejleszteni: mondja el, hogyan érti, mondja el a megoldását, indokolja a választott adatstruktúrát és algoritmust, vitassák meg az eltéréseket. A programozási nyelvet gyakorolni kell, mint a gépírást vagy az idegennyelveket. A géppel, illetve közvetlenül a fejlesztőkörnyezettel kell „kommunikálni”. Az elnevezéseket csak a kódrészlet humán értelmezéséhez kell megjegyezni, saját használatra. A gépnek mindegy, hogy a ciklusváltozó i vagy j, a programozónak kell tudnia, hogy melyik mit jelöl.

A programozás tanításakor nagyon figyelni kell arra, hogy a tanuló is tisztában legyen azzal, hogy különböző módokon kell megtanulnia a programozás egyes részterületeit. Oktatói részről folyamatos önkontrol kell arra vonatkozóan, hogy mit akar megtanítani, melyik területet akarja fejleszteni. Az első két pont gyakran úgy vetődik fel, hogy programozási elveket, modellezést (informatikát) tanítunk vagy nyelvet. A programozás oktatásnak [149, 150] egyik módszere a nyelv tanítása, nyelv-orientált. Bár sem az ELTE-n, sem a BME-n nem tartják jó módszernek, a tanulók egy része – mivel ez konkrét és magolható – lényegében nyelvet tanul. Informatikaoktatásban ezzel azonos szemléletű a menü oktatása. Ezzel szemben mind az algoritmus-orientált (ELTE) mind az adat-orientált (BME) a gondolkodásmód az informatikai értelmezést helyezi előtérbe, amelyhez csak eszköz a nyelv.

7.3.4

XI/4. Nagy beadandó

Mindkét megfigyelt tárgyban volt egyéni projektfeladat, bár nem így tartják számon, hanem „Nagy Beadandó” vagy „Nagy Házifeladat” néven. Nem követtem egyik egyetemen sem projektet – olyat, amit projektként tartanak számon –, de beszélgetések alapján úgy tűnik, hogy projekten olyan többórás csoportmunka értendő, amelynek van feladatmegosztás, tervezés, tanulás, végrehajtás és bemutatás része. Ezzel szemben azt gondolom, hogy egyéni projekt is lehetséges, sőt, a projektalapú oktatásnak fontos része. Az egyéni projektekben egy adott vagy választott feladat kell megoldani több lépésben. Ennek része a megoldás ütemterve, a hiányzó ismeretek megtanulása, a feladat elvégzése és a bemutatása, ami a munka és az eredmény dokumentálását is jelenti.

A BME VIK-en beadandó önállóan választott témájú feladata egy projektmunka, amiben a megvalósításra vonatkozó tartalmi és programozásmódszertani követelmények, valamint a dokumentálás formája kötött és a javasolt ütemterv betartására pluszpontokat lehet kapni.

Az ELTE IK-n a beadandó feladat egyedileg kötött, a dokumentációra erősen javasolt formátum érhető el, emellett ismert a pontozás részletes leírása, amiből következtetni lehet a tartalmi elvárásokra. Kicsit bizarr, hogy a tanított és a dokumentációban is elvárt sorrend – tervezés, modellezés, kódolás – helyett a program megírása és tesztelése az elvárt első lépés. Ennek oka az, hogy az összetett feladat megfogalmazása hétköznapi jellegű, néhány értelmezési kérdésre a tesztek adnak választ. A tervezést csak a feltételek, értelmezések pontos ismeretében lehet jól elvégezni. Így a projekt egy mintamegoldás elkészítésével kezdődik, ezt követi a szoftverkészítés és dokumentáció elkészítése.

Mindkét beadandóról, azaz projektről elmondható, hogy a tanultak LAU^{L}5b-5c ismereteit igényli, azaz a projekt során nem kell új tananyagot megtanulni, csak a korábban tanultakat kreatívan alkalmazni. A BME VIK projektje esetén a szabad témaválasztás jelentheti plusz ismeretek megtanulásának szükségességét – például grafikus megjelenítés választása esetén – ami gondot jelenthet, mert az érdekes és izgalmas feladat időben is többet igényel, és ez túlvállalást okozhat. Tovább nehezíti a teljesítést, hogy a határidő itt a szorgalmi időszak vége. Az ELTE-n több esetben tapasztaltam, hogy a nagy beadandó feladat elkészítése során nyernek összefüggéseiben is értelmet a tantárgyi követelmények.

Elképzelhető a tantárgyak projektalapú feldolgozása, amelyben a beadandók elkészítése lehetne a középpontban. A ZH-k súlyát jelentősen csökkentve, félévente két projektmunkával céltudatosabbá lehet tenni a tanulást. Nem a ZH-ra tanulna a hallgató, hanem a projektjének elkészítéséhez. A gyakorlatokon a feladatok megoldása azért szükséges, hogy a projektben fel tudja használni, a gyakorlat egyben konzultációs lehetőség is. Az a jó, ha minden gyakorlatról úgy áll fel a hallgató, az akkor tanultakat otthon beépíti a projektmunkájába (a beadandóba). Az előadásra azért járjon be a hallgató, hogy megtudja, hogyan készítse el a projektmunkáját. Így a projektekkel közeli határidős célt tűzünk ki, ami tanulási motivációt jelent, ezzel javítja a minőséget. Ez a fajta projektalapú oktatás oktatói oldalról szinte csak a mérés-értékelés módosítását jelentené.

7.3.4

XII. Mérés-értékelés

Minden képzésnek része a tanulók tudásának mérése, illetve a mérés eredményének értékelése. A közoktatásban a csoportot tanító tanár tervezi meg és állítja össze a dolgozatokat, ő értékel és az osztályozás is az ő feladata. Ezt egészíti ki az érettségi, illetve a kompetenciamérések.

Az érettségi pontosan leírt szabályok szerint zajlik, ismert a mérés tartalma, formája, időkerete – részenként is, többszörösen ellenőrzött a feladatok nehézsége, hogy összemérhetők legyenek a különböző években tett vizsgák és részletes értékelési útmutató biztosítja, hogy az adott vizsga értékelése egységes legyen. Informatika emelt szintű érettségi esetén a szabályozás 5% alatti (7 pontos) eltérést tesz lehetővé, amiből 1-2 pont az írásbeli javítási útmutató értelmezésében lehet – de ez fellebbezéssel is korrigálható –, 4-5 pont a szóbeli bizottság értékelésében a bizonytalanság, de ezt erősen csökkenti az, hogy a vizsgabizottságok tagsága változó és az eredményt a három tag konszenzussal hozza meg. Az 5 pont a tapasztalt eltérés a bizottsági tagok értékelésében az első feleletek során, azaz eddig a velem vizsgáztató bármelyik tag értékelésére az átlagtól legfeljebb ekkora eltérést tapasztaltam. Ez a későbbi feleleteknél 1–3 pontra csökkent. Az érettségi, mint kimeneti szabályozás vissza hat a helyi dolgozatokra mind a feladatok típusában, mind az elvárásokban, értékelésben, emellett helyben más típusú feladatokat is adnak a tanárok, az értékelésnek más szempontjai is megjelennek.

A kompetenciamérés teljesen más céllal történik, ez a mérés a tanuló számára nem értékelt, de az iskolát minősíti, az értékelés az iskolára, illetve az adott osztályra vonatkozik. Ez azt is jelenti, hogy a tanulónak nem érdeke se felkészülni, se másolni, viszont az iskolának érdeke felkészíteni a diákjait. (Egyes esetekben feltételezik, hogy a kitöltésben is segítenek a tanárok, de ezt én nem tapasztaltam.)

A fentieken kívül, az önértékelést támogatja a különböző versenyeken való részvétel. A tanórákon ajánlott folyamatosan visszajelezni a tanulóknak a munkájuk minőségét, egyes témaköröknél különböző formákban diagnosztikus mérések is szerepelnek – lehet ez gyors teszt, elkészített feladat bemutatása, házi feladat bemutatása.

XII/1. Mérés-értékelés jellemzői

Látható, hogy a közoktatásban van olyan mérés, amikor a diák képességeit mérjük, de nem a diákot értékeljük. Ilyen méréssel nem nagyon találkoztam az egyetemeken, pontosabban, a mérések értékelése elsősorban mindig a hallgatókra vonatkozik, a kapott eredményeket értékelik a szakra, tantárgyra vonatkozóan. Ebből adódik egy ismert, minősítéssel kapcsolatos dilemma: a szín ötös tantárgy esetén a hallgatók ennyire jól tudják az adott témát vagy munka nélkül

kapható a jeles, illetve, ahol sok a bukás, ott a hallgatók nem tanultak eleget vagy irreális az értékelés.

A közoktatásban a diák értékelése többnyire tanári kompetencia. Az egyetemi tantárgyaknál nagyon eltérő gyakorlatokat láttam. Az ELTE *Programozás* tantárgya hasonlított leginkább a közoktatásbéli módszerre. A beugrók kérdéseit és két kis ZH feladatait a tanárok állítják össze, a házi feladatokat és az évfolyam ZH-t a tárgy felelőse adja. A mérések egy részét gépi tesztekkel értékelik, másik részét a hallgatót tanító – gyakorlatvezető – értékeli közös útmutató alapján. A tárgyra kapott osztályzatot az egyes értékelésekből megadott súlyozással és ponthatárokkal számítják, de a tanárnak joga van az értékelést 1-gyel módosítani – lényegében dönthet úgy, hogy másik irányba kerekít. Az óraszámhoz viszonyítva az értékelések mennyisége nagy, ezért a tanárok a kötelezőn túl nem nagyon mérik és értékelik a hallgatók teljesítményét. A pluszban végzett értékeléseket a kerekítési szabályban tudják érvényesíteni.

A BME VIK *Programozás alapjai 1* és *2* tárgyban hasonlóak a szabályok. Minden mérés központi. A beadandó feladat kivételével az adott időpontban központilag összeállított feladatot kap a hallgató. A beugró a kis ZH és a házi feladat értékelése központi útmutató alapján a gyakorlatvezető feladata – itt még van esély a személyes visszajelzésre, de a nagy ZH javítása központi útmutató alapján központi javítócsoportban történik, az értékelésbe a gyakorlatvezetőnek nincs beleszólása. Az értékelés ilyen formában minimális tanári kompetenciát igényel, jellemzően szabályok követését igényli.

A központi javítás személytelen és – mivel az oktatók közül, aki ráér az vesz benne részt, továbbá kézírást kell értékelni – nagyon megterhelő. A teher csökkentésére egy beugróval szűrik a javítandó munkákat. A beugró – ahogy a laborgyakorlaton is – egy egyszerűnek szánt rutinfeladat, amelynek legalább félig jónak kell lennie ahhoz, hogy a feladat többi részét is értékeljék. A beugró arra a tapasztalatra épít, hogy aki egy egyszerű feladatban nagyon mást ír, az a feladat többi részében is össze-vissza irkál mindenfélét, amiből sok munkával lehet néhány pontot – vitathatóan – adni. Az indoklás elég szubjektív. Amikor átnéztem a munkákat és az értékelést, ez a szempont nem tűnt valósnak, mert a néhány adható pont megtalálása nem okozott nagyobb gondot a gyenge beugrók után, mint egyes jó beugrók után. Ezzel szemben, volt olyan dolgozat, ahol a teljesen félreértett beugrót követően tiszta, szép, majdnem jó megoldást olvastam, illetve a hibásan megoldott beugró után semmit nem írtak.

Az, hogy 10 pontból az első két pont egyikét meg kell szerezni, hogy a többi nyolcpontnyi részt is értékeljék, elég nagy reklamációs rizikó, ezért a beugrón elbukó feladatokat a javításvezető – itt egyben tantárgyfelelős – is átnézi, felüljavítja. (Láttam olyan dolgozatot, aminek a

beugrója 0 pontos volt, az eredmény mégis 4 pont lett és ezzel az összpontszám is elérte az elégséges szintet.) Azaz, pont azokat a feladatokat értékelik duplán, amikkel nem szeretnének foglalkozni. Emiatt azt gondolom, hogy a beugróval szűrés összességében nem csökkenti a javítási munkát, viszont a hallgatókat stresszeli, igazságérzetüket borzolja és egyes feladatoknál a feladat szövegében a kétpontos rész leválasztása a feladat megértését nehezíti.

A BME VIK Adatbázisok tárgyát is megfigyeltem 2013-2015-ben. Ezt a tárgyat sok demonstrátor oktatja. A feladatsorok összeállításában is részt vesz minden oktató és a javítást is közösen végzik. A szemeszter végén írásbeli és – ritkaság, de – szóbeli vizsga is van. Az írásbeli a beugró, ami négy kérdésből áll. Ezek gyakran eldöntendő kérdések, ahol a választ magyarázni is kell (Igen/Nem? Miért?), összesen 8 pontot lehet elérni, négy ponttól mehet az aznapi szóbelire a hallgató. Ez a beugró tényleg szűr, sokak szerint túlzóan. Valójában nem a gyenge elégteleneket szűri ki, hanem körülbelül a közepes szintig mindenkit. A kérdések gyakran a definíciók, összefüggések legapróbb részletére kérdeznek rá – amihez nem elég tanulni, de érteni is kell a tananyagot. A vizsga érdekessége, hogy a beugrót „meg lehet védeni". Az Igen/Nem válasz néha „Is", azaz „úgy is lehet érteni, és akkor…" Ehhez azonban a hallgató magabiztossága szükséges. A beugrót követően a szóbelin az bukik meg, aki „a legpechesebb" tételt húzza, de ezekre jellemző, hogy meg sem próbálja a feleletet, így a szóbelin többnyire közepes, jó vagy jeles osztályzatok születnek – ami a beugró után szinte mindegy. A vizsgáztatásban viszont a demonstrátorok is részt vesznek. Azaz a gyakorlatvezető egy erős előszűrő után önálló minősítő.[75]

A közoktatási és ELTE-s tapasztalatommal ellentétben a BME-n azt tapasztaltam, hogy a hallgatót oktató személynek minimális felelőssége van az értékelésben. Az értékelés a tantárgyfelelős kompetenciája, ez biztosítja az egységességet.

7.3.5

XII/2. A mérés minősége

A dolgozatok mindegyike a tanultak alkalmazásának rutinját méri, elvileg LAU^{L}5a szinten, azaz a tanultakat abban a formában kérik számon, ahogy az az előadáson, gyakorlatokon szerepelt. Azonban a feladatok valamilyen gyakorlati környezetbe ágyazva jelennek meg, aminek értelmezése okozhat problémát. Egyrészt nagyon nehéz olyan témát választani, amelyiknek a

[75] Háttér: 2014-es vizsgaidőszak „beugró" kérdéseinek és a válaszoknak az elemzése; a beugró és a szóbeli között a beugróra vonatkozó kérdőívre adott válaszok, megfigyelőként részvétel a szóbelin. Máshol nem publikált adatok.

fogalmait mindenki érti. Másrészt a feladatok szövege illeszkedik a számonkérendő témához, ami miatt a feladat néha erőltetett, az absztrakció nem fedi a tapasztalatot.

Példa rosszul értelmezett feladatokra:

- A pixel színének megadása három számértékkel esti tagozaton nagyon sok hallgatónak ismeretlen volt, a három számot három pixel színeként értelmezték. Az ismeretet a közoktatásból kellett volna hozniuk, de ez az adott hallgatók esetén esetleges volt.
- Kutyák listáját kellett elkészíteni, amiben egy-egy kutyáról a szülők azonosítóját is tárolni kellett. Nagyon sok hallgató az azonosítók alapján bináris fát (családfát) akart készíteni, bár a feladat szövegében többször szerepelt a lista szó. Úgy tűnik, az egyed-szülők adataiból készített dinamikus struktúra kognitív képe felülírta a feladat szövegét.
- Kávékból kellett heterogén kollekciót készíteni, kávéfajta lehetett Presszó, Java és Irish, az első kettő lehetett arabica és robusta tulajdonságú, az Irishnek szám típusú tulajdonsága volt. Ezeket a kávékat termoszba kellett tölteni, ami ott sort képezett. A megoldás titka: ne akard megérteni, hogyan lesz sorban a termoszban a kávé, hogy miért alternatíva a presszónak az ír kávé…

A feladatsorokat rendszeresen ellenőrzik, előre megoldják, de jellemző (főleg a BME-n), hogy az értelmezési problémák leküzdésére szánt idővel rosszul számolnak. Az ELTE-s ZH-k után nem hallottam „kevés volt az idő” véleményt, viszont a BME-n rendszeresen előjött, hogy az egyik feladatnál elakadt és ezután a többi feladatot is elkapkodta.

A megoldás sikerességét befolyásolja az is, hogy az ELTE-n a hallgató kérdezhet. Itt a ZH része a feladat értelmezése, ezzel kapcsolatban bizonyos kérdéseket feltehetnek. Azonban ez a lehetőség egyben a visszaélésre is lehetőséget ad, ugyanis a kérdés elhangzása előtt nem tudható, hogy hogyan fog szólni, az elhangzó kérdés viszont segíthet a szomszédoknak. Például: „Az összes azonosítószámot ki kell gyűjteni, vagy csak meg kell számolni a megfelelő azonosítók számát?” – a feladat első része, hogy meghatározzuk, melyik tételről van szó, a kérdés tartalmazta a választ: kigyűjtés vagy megszámlálás. A mérés (dolgozat) közben engedélyezett kérdés további problémája, hogy eközben az oktató figyelmét leköti a kérdező, így a többinek lehetősége nyílik a kooperációra, puskázásra.

A BME-n a kérdezés nem jellemző, a hallgatók nem kérdeznek rá a félreérthető fogalmakra. De egy félreértésből adódó más típusú megoldás lehetetlenné teheti a mérést, és mindenképp befolyásolja a megoldás idejét.

Minden dolgozatnál az adminisztráció – a vizsgán megjelentek adatainak egyeztetése – felügyeleti lazaságot jelent. A számítógépes vizsgák esetén ez kisebb gondot jelent, mert a bejelentkezés adja az azonosítás alapját, az online kommunikáció és a többszörös bejelentkezés tiltható, a mobil telefonnal a monitort kellene lefényképezni, amit nem könnyű testtel kitakarni. Ezzel szemben a papíron írt dolgozatoknál az adatok ellenőrzése kitűnő alkalom a feladatok fényképezésére, elküldésére. Más karról részletes történeteket tudok a feladatok háttérben történő megoldásáról is, megosztott megoldásáról is. A mérés hitelességével kapcsolatos az is, hogy a felügyelők az írásbeli vizsgán jellemzően szemből figyelik a munkát, ami a pad alatti tevékenységeknek széles teret ad. Az ELTE-n felügyelt vizsgán figyeltem fel arra, hogy azok a felügyelők, akik középiskolában is tanítanak mind a terem hátulját részesítik előnyben, mert ebben az esetben a hallgatónak speciális mozgásra van szüksége – például ellenőrizni a felügyelők helyzetét – ahhoz, hogy mérésen kívüli tevékenységet végezzen. A számítógépes vizsgák esetén is a felügyelet – monitorral vagy személyes jelenléttel a hallgató háta mögül, a képernyővel szemben célszerű.

7.3.5

XII/3. A mérőeszköz

Szerettem volna megvizsgálni, hogy az informatika, illetve programozási ismeretek mérésére milyen eszközök, formák alkalmasak. A domináns mérési mód a gyakorlati feladat – matematikából szöveges feladatnak felelne meg – megoldásának kérése megadott feltételek mellett. Ugyanazt a feladatot többféle absztrakcióval is fel lehet dolgozni. A közoktatásban tipikusan ilyen a dokumentumkészítés – például tételkidolgozás – szövegszerkesztővel, prezentációként és weblapon vagy adatelemzés – például Európa országairól – táblázatkezelő, adatbáziskezelő és programozási feladatként. Az egyetemi programozásoktatásban is felhasználhatjuk ugyanazt a feladatot többféleképpen: C-ben statikus tömb, dinamikus lista és heterogén kollekció is lehet egy fehérjelánc. A vízállásjelentés adataiból több tucat algoritmizálás feladatot lehet generálni.

Az egyes megoldási környezeteket le lehet szűkíteni a felhasználható szoftverek, alrendszerek, rendelkezésre álló eszközök, függvények megadásával, az internet használat engedélyezésével vagy letiltásával. Például lehet-e használni a C++ STL könyvtárát, C# LINQ lehetőségeit vagy Lambda kifejezéseit, lehet-e C-ben bool típus. Úgy tűnik, az informatika specialitása, hogy a lehetőségek szűkítése jellemzően a kreativitás nagyobb elvárását igényli. A „mindent szabad használni” matematikából, azt jelenti, hogy a feladathoz kell egy-több ötlet. Ugyanez informatikából inkább arra ösztönöz, hogy megnézzük az interneten, hogy van-e a problémára

kész megoldás. – Nemsokára talán azt is mondhatjuk, hogy van-e MI, ami megoldja helyettünk a feladatot.

A Nemes Tihamér Nemzetközi Informatikai Tanulmányi Verseny Programozás kategória és az Informatika II OKTV első fordulójában – amit papíron írnak, de semmi akadálya nincs a gépesítésnek – több más típusú feladatot is megtalálhatunk. Például hibakeresés; mit csinál?; programkiegészítés; néhány speciális nyelvi elemmel megfogalmazandó feladat. A BME VIK *Programozás alapjai 2* tárgy kis ZH-ban tesztkérdések is szerepelnek. Bár sokféle feladattípus alkalmas lehet a mérésre, ezeket a feladattípusokat csak nagyon kis mértékben alkalmazzuk.

- A hibakeresés nagyon rizikós mérési eszköz, mert többféle lehet a hibák kijavítása, ami nehezíti az egységes értékelést.
- A mit csinál? hasznos lehetne, de 1-1 feladat sok oktatói előkészületet és kevés hallgatói munkát igényel.
- A program kiegészítés szintén az előkészítői munka miatt lehet felejtős – írja meg a programot teljesen a hallgató –, de az automatizált, MI-val támogatott oktatásban jelentős lehetne a szerepe.
- A speciális nyelvi elemekkel operálás lényegében egy másik programozási nyelv alapjait jelenti, általában nem fér bele a tananyagba.
- A tesztkérdések csábítónak tűnnek, mérésre jók, de az értékelés nagyon sok problémával jár. Az informatikában tipikus, hogy adott kifejezéseknek a környezettől függően más a jelentésük, változik az értelmük. Önmagában, ahogy azt az Adatbázisok beugróinál is láthattuk, az „Igen/Nem” válaszok mellett szükséges egy „Miért?”, számítani kell arra, hogy a válaszok az értelmezési domaintől függően változhatnak. Ezért – bár nagyon csekély volt a minta, – tesztkérdéseket csak feladatbankban, előzetesen gyakorolhatóan tudok mérés során elképzelni.

7.3.5

XII/4. Az értékelés

Az ELTE-n a vizsgák fele, a BME-n minden vizsga papíron történik. Ez nemcsak a vizsga hitelességét gyengíti, hanem jelentősen megnehezíti az értékelést. A gépen írt vizsga egyik ellenérve, hogy ha fejlesztőkörnyezetben írják a vizsgázók a dolgozatot, akkor sok olyan probléma is felvetődik, amit a feladat nem kér. Valóban, könnyebb „cifrázni” a programot, egy pontosvessző hiánya megakadályozhatja a futtatást. Ugyanakkor a papíron írt vizsgákról kimutatható, hogy a tananyagban szereplő tesztelés, számítógéppel történő hibakeresés (debug használata), konkrét programkészítés készségét nem mérik, a tananyag egy fontos szegmense nem

értékelt. Ezzel szemben, a számítógépen írt vizsgák esetén lehet automatikus ellenőrzést, tesztelést végeztetni. Ebből azonban nem következik, hogy egy gépen írt vizsgához a fejlesztőprogram használata lenne szükséges. A vizsgákhoz egyszerű űrlapos rendszerek vagy egy szövegeditor használata is elég lehet, a digitálisan tárolt válaszokra lehet kiértékelő programot, akár mesterséges intelligencia alkalmazást készíteni. Mivel a dolgozatok LAU^{L}5a-5b szintet mérnek (mást nem tudnak), a válaszok eléggé sablonosak kell legyenek, ezért egy plágium ellenőrző a mintamegoldás és a jó megoldások között nagyon nagy egyezést kell mutasson, egy dokumentumösszehasonlító szoftver az eltérések színes kiemelésével segítheti a javítást – és ehhez lényegében szoftver fejlesztés sem kell.

A papíron írt kód lineáris írásmódjával a magolást támogatja, egy gépen írt kódot lehetne lépésenként tárolni, így az is mérhető és értékelhető lenne, hogy a kód blokkos szerkezetét blokkonként írja-e a vizsgázó. A vizsgán kért feladok megírási algoritmusát is lehetne értékelni, egyes esetekben akár animációként le lehetne játszani a kód elkészítését.

A dolgozatok értékelése, a papíron írtaké is, lehetne elektronikus. Az egyes itemekre adott értékeket táblázatba lehetne gyűjteni, ami lehetővé tenné a tantárgy- és mérésfejlesztést is, mert láthatóvá válna évfolyam szinten itemre bontva a sikeresség. Ez a megoldás a közoktatásban az informatika érettségire, az alkalmazói versenyekre elektronikus mérés mellett, a középiskolai felvételire papír alapú mérés mellett működik, bár a felvételinél az adatfelvitelt nem a javítótanár, hanem adminisztrátor végzi. Az ELTE programozás tantárgyában a papír alapú központi vizsgákhoz elektronikus javítókulcsok készülnek, a gépes vizsgán az értékelést feketedoboz módszerrel a gép végzi. A BME-n megfigyelt javítások piros tollal, a papíron történtek. A gépesítés ellen – úgy tűnik –, számos érvet tudnak sorolni. Az oktatók könnyedén tudják az ujjukon 10-ig jegyezni a pontszámot, a papíron jelölik, hogy mi a hiba, néha megjegyzést is írnak hozzá – ezért a toll a kezükben van, rögtön pontozhatnak is. Az A4 méretű nyomtatott pontozási útmutató, dolgozatlap, javítandó és már kijavított dolgozatok kupaca mellett útban van a laptop, kényelmetlen a plusz eszköz használata.

Az elmúlt 17 évben a dolgozatok 90%-t vagy „ránézésre”, ötfokú skálán értékeltem, vagy itemenként, elektronikus táblázatban pontoztam. Régen, matematika-fizika szakos tanárként pontoztam papíron, de a ZH javításokon újra kipróbáltam. Az ujjamon számlálás csak akkor megy, ha a feladat közben senki sem zavar meg, nem kérdeznek. Ez a csoportos javításnál elég ritka, ezért volt, hogy háromszor pontoztam egy feladatot. A papírra jegyzett részpontok vagy pontlevonások hatékonyabbak, de sokszor más sorrendben van leírva a megoldás, mint ahogy a pontozási útmutatóban szerepel, ezért a pontszámnak többféle helye lehet, nem látszik, hogy

mit értékeltem és mit nem. Hasonló problémát jelent a levonó értékelés – a hibát jelzem és írom a levonást, de ez sem egyértelmű. A megadott vagy levont ponthoz tartozó szöveg részletes leírása nagyon lassítja a munkát. Én egy idő után vittem gépet, a javítást a táblázat elkészítésével kezdtem és tároltam a pontokat. Ez lehetőséget adott arra, hogy a sokadik javítás után is, ha kiderült, hogy egy hibát rosszul súlyoztam, vissza tudtam keresni azt a néhány dolgozatot, amiben ez előfordulhatott.

Számomra aggasztó, hogy a programozás tantárgyat mérnökinformatika szakon papíron írják és értékelik, miközben az oda felvett diákok a közoktatásban ugyanilyen dolgozatokat gépen írnak és elektronikusan kapják az értékelést. Nemzetközi kitekintésben pedig egy Hollerith előtti módszer alkalmazását látom, miközben Kínában 5 év múlva robotokkal akarják megszervezni az oktatást.

7.3.5

XII/5. Az értékelés fejlesztő hatása

A dolgozatok, mérések és értékelések alapvetően két célt kellene, hogy szolgáljanak. Az egyik a teljesítmény minősítése, a követelmények teljesítésének igazolása, a másik visszajelzés a hallgató felé, a fejlesztendő területek meghatározása. A pedagógiában a második szempont a fontosabb, de az egyetemeken erre nagyon csekély példát találtam. A „csak görbüljön” elvet az oktatók sem szeretik, igyekeznek a feltételekkel a lehető legszűkebbre venni az elégséges szintet, de a közepes-jeles skálán sokak szerint a szerencse is számít. Amikor oktattam, külön figyelmet fordítottam arra, hogy csak a tökéletes munkát fogadjam el, ösztönözzem a hallgatókat az apró hibák kijavítására is. Ez sokszor nagy erőfeszítést igényel a hallgatótól, ami később a méréseken esetleg nem látszik. A – saját elvárásához képest – alul-teljesítés erős demotiváló hatású. Ehhez társul szinte minden vizsgált tárgynál, hogy a dolgozat megtekintése nem támogatott, a dolgozatban elkövetett hibák elemzésére nem kerül sor.

A dolgozat megtekintésére kötelező lehetőséget biztosítani, de a megtekintés lehetősége, ideje, módja annyira abszurd, hogy mindenki tudja, csak akkor érdemes odamenni, ha nagyon fontos javítani a pontszámon.

Példa megtekintés szabályára:

Egyik esetben, a gyakorlaton lehetett a ZH feladatot megtekinteni, ahol a következő szabályok voltak:

- tollat, ceruzát, fényképezőgépet nem hozhatsz;
- ellenőrizd, hogy ki van-e javítva minden feladat, de biztos, hogy igen;

- ellenőrizd, hogy jól adták-e össze a pontjaidat, de nem szokott hibás lenni;
- kaptál 2 pontot pluszban, ha úgy látod, hogy a jelenleginél több pontot ér a dolgozatod, akkor kérheted az újrapontozását, de akkor azt a 2 pontot elveszíted;
- egy percet kapsz a megtekintésre.

Ilyen feltételek mellett néhányan megkérdezték, hogy ha kérdeznek, akkor is elveszítik-e a 2 pontot. A kérdés a javításba írt jelölésre, megjegyzésre vonatkozott volna, azt nem érette a hallgató, de ez reklamációnak számított volna, mert a helyes megoldás a neten megvolt, előre megnézhette volna, megtanulhatta volna.

Megvizsgáltam, hogy a ZH megírása után közzétett pontozási útmutató és (egy) helyes megoldás mennyire használható a hiányok pótlására, fejlesztendő területek kijelölésére: csoportomból az alul teljesítők dolgozatát kikerestem, lefényképeztem és elküldtem a hallgatónak, hogy írja meg kijavítva és kommentben jelölje, hogy mi volt a gondja, mi volt a hiba a ZH alatt. Volt, aki visszaírta, hogy a feladattal nem érdemes foglalkoznia, mert időhiány miatt nem tudta átgondolni, csak írt valamit. De volt olyan is, aki megírta, hogyan látja a helyzetét.

Példa a ZH értékelésének értelmezésére:

Az alábbiakban a hallgató kijavított munkáihoz írt megjegyzéseit írom ki és ezeket minősítem.

- „(char *str, char c) egy str sztringben kerestem volna az adott karaktert, és akkor térjen vissza, amikor megtalálta. A javításban az szerepel, hogy ha megtalálta return false?” – A megoldási útmutatóban egyetlen karakterről kellett volna eldönteni, hogy tagoló-e, később ezt a függvényt kellett volna minden karakterre meghívni. A dolgozatban egy-egy tagolókarakter meglétét vizsgálta volna a stringen belül. Hiba: a beugró feladat félreértése. Bár a vizsgázó gondolatmenete is jó lehetne, de nem ezt kérte a feladat. A különbséget nem érti, az ő gondolatmenetét viszont nem értékelik.
- „Nem tudom mit jelent a kódduplikáció, talán, hogy sok feltétel nem lehet egymás mellett? A megoldást értem, csak azt hittem úgy is meg lehet oldani, a sok egymás melletti feltétel miatt nem lehet?” – A „kódduplikáció” piros jelölés azért szerepelt, mert a beugróban hibásan megírt függvényben ugyanaz a feltétel szerepelt, mint a folytatásban. Azon a ponton nem a sok feltétel, hanem a jól megírt beugró feladatra hivatkozás hiánya okozott gondot. A hallgató nem érti a javítást, nem érti miért nem fogadható el a munkája, csak annyit lát, hogy másképp kell.
- „Nem értem miért rossz típus ez a kettő, strcmp-vel lefutott hiba nélkül.” – A két típus, amire hivatkozott, jó volt, de pótlólag írta be és nem fért a sorok közé, ezért a

felhasználás után írtnak látszik. A feladatban definiált függvény visszatérési típusa rossz, mert bool logikai értéket írt a megoldás többi része alapján, void helyett. – Jól látható, hogy nem érti a javítás jelöléseit, nem látható, hogy a megjegyzés mire vonatkozik.

A folytatásban még 2 félreértés, több „időhiányban ezen nem gondolkodtam”, 2-szer „ezt nem tudtam” – strcmp, rand – és 1-szer nem jutott eszembe, hogy így lehetne – tömb használata – szerepelt. Több helyen olvasható, hogy „a megoldást értem, de magamtól elég sok idő lenne, mire sikerülne”.

A ZH-t ezután megbeszéltük, de az idő szorításával nem tudtunk mit kezdeni. A hallgató más szakon MSc-t végzett, ezért a ZH eredményét (eredménytelenségét) látva felmérte, hogy nem lesz ideje a sikeres befejezésre.

A közoktatásban a dolgozat utáni órán meg szokás beszélni a feladatokat, a dolgozat értékelése és javítása nem az eredmények ismertetése, hanem a hiányok pótlását, hibák megbeszélését, javítását jelenti. Az oktatónak nem az az elsődleges feladata, hogy betöltse a tudást a hallgató fejébe, mert azt a hallgató önállóan is meg kell tudja szerezze, hanem az, hogy segítsen tanulni a hibákból, problémákból. Ezért fontos az *ellenőrző feladat* is, de nagyon fontos lenne a ZH-k elemzése, a tanulási feladatok meghatározása.

Az értékelés fejlesztő hatását tovább rontja, hogy a javító dolgozatok – bár kötelezők – nem biztosítják a megfelelő fejlődést. Általában egy témából két, egymásra épülő dolgozatot írnak a hallgatók. Az, hogy a dolgozatok anyaga egymásra épül, azt is jelenti, hogy nem érdemes addig a második dolgozatnak nekifutni, amíg az első, a könnyebb, nincs meg. Először be kell hozni a lemaradást, ezután érdemes csak továbblépni. A javító dolgozatok azonban – oktatói kapacitáshiány miatt – ezt nem teszik lehetővé. Azt gondolom, hogy a dolgozat értékelését fejlesztésre is felhasználva, az első dolgozaton elbukóknak felzárkóztató feladatok kellenének és legkésőbb a második dolgozat idejében kellene megírni az első dolgozat javítóját, majd ezt követően, legkésőbb a javítások idején tehetnének próbát a második dolgozat megírására, sikertelenség esetén egy héttel később lehet ennek javítása. Nem véletlen, hogy a Programozás alapismeretek 1 tárgyból a javításra jogosultaknak csak 53%-a ment el javítani 2017-ben.

7.3.5

XII/6. A nem fejlesztő értékelés

Azokban az esetekben, amikor az értékelés nem ad fejlesztési célt, az egyetlen cél a minősítés, a pontszerzés marad. Ezt a hallgatók tudomásul veszik, ennek megfelelően lépnek fel. A meg-

tekintést csak azoknak érdemes kérnie, akik az eredményükön javítani szeretnének. Mivel megbeszélésre, megértésre nincs mód, marad az igény bejelentés: kellene még n pont. És elindul egy olyan kommunikáció, ami az oktató tudása helyett a jóindulatára próbál hatni, ami megkérdőjelezi az oktató szakmai önbecsülését. Ez a pontszámért való alkudozás az, ami miatt a megtekintéstől távol akarnak maradni az oktatók.

2017-ben a Programozás alapjai 1. dolgozatok megtekintésére új eljárást alkalmaztak. Ez dolgozat egyéni kiértékelését motiválja, így a hallgató számára fejlesztő, ugyanakkor a pontszámért való alkudozást kizárja. A módszer lényegében az érettségi reklamáció hivatalos útja, ebből következően a hallgató és a pontozás felülbírálásáról döntő személy nem kerül közvetlen kapcsolatba, ugyanakkor a hallgató oktatói, gyakorlatvezetői, ismerősei tetszőleges formában segíthetnek a hallgatónak megérteni az értékelést és – ha úgy gondolja – kérni a pontozás módosítását. A lényeges pontok: megtekintéskor a kijavított papíron írt dolgozatot a hallgató lefényképezheti, gépen írt dolgozathoz olvasási hozzáférése lesz, a pontozást a legrészletesebb módon megismerheti. Ezen források alapján tételes pontszám módosítási kérelmet írhat, amelyben pontosan megjelöli a felülbírálandó helyet, kapott és nem megkapott pontokat és leírja, hogy miért gondolja, hogy a megoldását másképp kellene értékelni, szerinte hogyan kellene értékelni. A javítás felülbírálója a kérelem alapján áttekinti a jelzett részt és eldönti, hogy a kérelem mennyiben jogos, újra értékeli az adott részt. Az eredményről értesíti a hallgatót.

Természetesen ebben az esetben is lesznek megalapozatlan pontszámnövelő kísérletek, de ez nem személyes könyörgés, korrumpálás formájában jelenik meg, hanem egyszerű, eldöntendő kérdésként, ami a javítónak csak a szakmai kompetenciáját érinti, a lelkét, empatikus készségeit nem.

7.3.5

XIII. A programozási nyelv

Az ELTE IK-n és a BME VIK-en is említették oktatók, hogy a legjobb tanulónyelv – első programozási nyelv – szerintük a Pascal. A nyelv szintaktikája segíti az imperatív, procedurális gondolkodásmód kialakulását, a módszeres programozás és az OOP alapjainak megértését is. Azonban a nyelvben a begin-end feleslegesen hosszú, ezért nincs jelentős folytatása a szakmában. A C szintaktikáját átvevő nyelvek uralkodnak. Ezért az imperatív nyelveket C nyelvre alapozva célszerű tanulni, abból származnak vagy következnek az egyes nyelvek egyedi tulajdonságai. A nyelvek készítői a C-nyelv definíciói vagy azok alternatívái alapján hozzák létre a „C alapú nyelveket”. De a C-nyelv nehéz, túl gépközeli, túlságosan hamar szükséges az indirekció, a pointer.

A C-nyelvre épül, annak kiterjesztése a C++, amit tudatos tervezéssel sokféle feladatra továbbfejlesztettek, de a gyökerek megtartása miatt egyre összetettebb a szintaktikája. A C# viszont az alapoktól definiált nyelv, ami a C bizonyos előnyeit próbálja ötvözni az egyszerű használhatósággal. A Java nyelvet az OOP nyelvének szokták nevezni… és lehetne sorolni a többi, most éppen divatos vagy feltörekvő, vagy hanyatló népszerűségű nyelveket és egyediségüket jellemző tulajdonságaikat.

A programozás oktatásában másodlagos szerepet játszik az oktatott nyelv. Az elsődleges szempont a programozás informatikai aspektusa: a programkészítés elvei, paradigmái, a problémamegoldási módszerek. A nyelv csak kifejező eszköz. Ezért – úgy vélem – a programozás oktatásához mindig olyan nyelvet kell választani, amellyel a tanítani szándékozott elvek, módszerek a legoptimálisabban megvalósíthatók, bemutathatók. Ennek megfelelően, a géppel való szofisztikált kommunikációhoz szövegalapú programozási nyelv illik; az adatok és objektumok tanulmányozásához erősen típusos nyelv való. Ugyanakkor a programozás tanulását a mindennapi jelenségek oldaláról kell kezdeni, induktív módszerekkel, ahol a pontos részletek homályban maradása természetes.

A leendő szakembereknek a programozást az alapjaitól, nulla szintről kell tanulnia, de ezt meg kell előzze a felhasználási szintről az alapokig az áttekintés, a tapasztalatszerzés. Ez tükröződik az „első programozási nyelv előtti programozási nyelv” tanításában.

Bjarne Stroustrup egyetemén, a Columbia University-n egy hathetes, 12 óra előadást tartalmazó előkészítő[76] kurzuson Python a bevezető nyelv. Sok más egyetemen is, ahol nem számítanak arra, hogy a felvett hallgatók már tanultak programozni, bevezették, hogy először Pythont tanítanak.

David J. Malan a Harvard College tanára, aki a CS50[77] tárgyat tanítja több száz hallgatónak, valamint edx-en MOOC formában milliós nagyságrendű külsős követőnek. Malan a C-nyelv előtt a Scratch-et használta bevezető nyelvnek, a második héten tér át a C-re[78]. 2018-as tanfolyamán[79] már megjelent a Python is, de nem bevezetésként, hanem mint egy más szintaktikájú, egyszerű – vagy legalábbis más – kódolású divatos nyelv.

A University of Cambridge oldalán ezt olvashatjuk:

> *„If you choose to learn a new language, it may be a good idea to learn one that is not explicitly taught in the Tripos. Doing so obviously helps to avoid repetition, but also gives you a wider perspective on languages that can be useful later in the degree and in employment. A popular choice is Python, for which there are many tutorials available."*[80]

Az első évben tanított Foundations of Computer Science oktatásához az ML funkcionális nyelvet használják, ezzel párhuzamosan tanítják az Objectum-Oriented Programming tárgyat Java nyelven.[81] Az ML nyelvet – a kurzusleírás alapján – már az első héten használják, míg a Java használata előtt elméleti órák, objektumtervezés, UML szerepel. Azaz a Java követi az ML-t.

Jól látható az is, hogy az egyetem elvár valamilyen programozási gyakorlatot. Az idézet mellett az is olvasható, hogy ez lehet bármilyen programozási nyelv ismerete, mobil applikáció fejlesztése, robotika.

Az angol közoktatásban már bevezették a programozás tanítását, de ez nem feltétlenül tükröződik a felvettek körében, ezért ajánlja a Pythont is. Az amerikai közoktatásban nem kötelező programozást tanulni, ott a teljesen kezdőkre is készülnek. Ezért szerepel az első alkalmak egyikén a Harvardon a Scratch, ezért indít Pythonnal a Columbia University.

---

[76] http://www.cs.columbia.edu/~bauer/cs3101-1/ [2021.10.30]
[77] https://online-learning.harvard.edu/course/cs50-introduction-computer-science [2021.10.30]
[78] https://www.youtube.com/watch?v=u-kH-5JJSgU [2021.10.30]
[79] https://www.youtube.com/watch?v=hnDU1G9hWqU [2021.10.30]
[80] https://www.cst.cam.ac.uk/admissions/undergraduate/faqs [2021.10.30]
[81] https://www.cl.cam.ac.uk/teaching/1920/cst.pdf [2021.10.30]

A BME VIK-en C az első nyelv. Jól láthatóan nehéz azoknak, akik más nyelvet még nem tanultak. A közoktatási informatika a legújabb változatban, ami 8 év múlva érvényesül az egyetemen, van programozásoktatás. Addig fakultáción vagy egyéni úton tanulnak valamilyen nyelvet a diákok. Amennyiben ezek az alapok megvannak, a képzés profiljának, az adatorientált programozásoktatásnak a C felel meg leginkább. El lehet gondolkodni azon, hogy a C++ sokkal több helyen használható, át lehetne-e térni erre a nyelvre, de ehhez a döntéshez feltétlenül figyelembe kell venni, hogy a villamosmérnökök is azt tanulják, mint a mérnökinformatikusok. Nekik is jobb lenne a váltás? Érdemes más nyelveket oktatni a villamosmérnököknek, mint a mérnökinformatikusoknak? Másik eldöntendő kérdés, hogy a C++-ra áttéréssel továbbra is adat-orientált módon kellene tanítani a programozást – a C++-ból a C hagyományos részének megfelelő subsetet kellene használni –, vagy jobb lenne objektum-orientált programozással kezdeni mindjárt az első félévben.

2018-ban indult a BME VIK-en a BProf, üzemmérnökinformatikus képzés. Itt a Python az első nyelv. A pontos indokot nem ismerem, hallomásból a nyelv könnyűsége és divatja tűnik a választás alapjának. Alapfeltevés volt, hogy a szakot megkezdők előtte nem tanultak programozni. Kérdés, hogy ez igaz-e. További kérdés, hogy mit kell Python nyelven megtanítani. Bevezető nyelvként csak a legegyszerűbb dolgokat néhány probléma feldolgozásával (problémaorientált esetleg kompetitív oktatás) vagy mélyebb adat és algoritmus ismeretek szükségesek? Továbbra is adatorientált oktatás lenne, de egy objektumorientált nyelven? Vagy az objektum-orientált szemléletet vezeti be, mert az üzemmérnök-informatikus nem gépközeli kódolással, beágyazott rendszerekkel foglalkozó programozó lesz, hanem mesterséges intelligenciákat fog fejleszteni?

Az BME VIK-en és az ELTE IK-n is azt hallottam érvként a Python mellett, hogy más egyetemek is Pythont oktatnak bevezető nyelvként, mert az könnyebb, bent tartja a hallgatót. Nincs elég példám, de az eddigiek azt mutatják, hogy a Python a „más egyetemeken” egy felhasználói alkalmazás, részkészségek fejlesztésére használják, bevezetésként. Emellett van az „első programozási nyelv”, amiben a megfelelő programozási paradigmákat tanítják. A Python helyettesíthető Scratch-csel, oktatásuk a képzés elején nem a szakemberképzés része. Ebből annak kellene következnie, hogy ha a közoktatásban teljesértékű lesz a programozásoktatás, akkor az egyetemen nem lesz szükség bevezető nyelvre így a Python elsőként tanítására sem.

Az ELTE IK-n az algoritmus-orientált módszeren van a hangsúly, a nyelv kevésbé fontos. De mégis nagyon sokat számít. A programozás alapismereteinek oktatásához három nyelvet

használ: matematikait a specifikációhoz, struktogramot a modellezéshez és C++ alapokat a kódoláshoz. Ezzel párhuzamosan a programozás gyakorlati (adat, objektum, fordítás) tulajdonságainak megismerése C és Python nyelven történik az Imperatív programozás tantárgyban, Haskell-t tanítanak a Funkcionális programozás tantárgyban, PowerShell-t és Bash script használatát a Számítógépes rendszerek tárgyban. Ez nyolc nyelv, egy félév alatt. A specifikációt és a struktogramot azért számítom külön nyelvnek, mert más a szintaktika. n∈ℤ után n:Egész majd int n;… az adott kód minden sorát le kell fordítani a következő absztrakciós szintre. A Python azért szerepel a listában, mert a második félévben a Diszkrét matematikát oktatók ezt szeretnék használni. A PowerShell és a Bash a Windows-Linux egyenrangú oktatása miatt szerepel párban. A Haskell azért kell, hogy ne csak az imperatív gondolkodást ismerjék meg a hallgatók, hanem a deklaratívat is… Az érvek érthetők, de számomra hasonlít a mesebeli mindentbeletortára[82]. A C++ tárgyban az egyik hallgató C kódot ír csillagokkal, malloccal; a másiknak nem tetszik, hogy „tömb” az adatstruktúra neve, szerinte listának kellene nevezni. Sokszor megállnak kódolás közben, hogy átgondolják, az adott esetben hogyan írják szintaktikailag helyesen a kódot. Félév végére a Pythont el is felejtik, mert a C fontosabb.

Természetes, hogy egy programozónak tudnia kell újabb és újabb nyelveket tanulnia, abban dolgozni. De kezdetnek – valószínleg – jobb lenne fele ennyi nyelv, annak alaposabb ismerete.

XIII/1. Nyelvismeret mélysége

Az University of Cambridge képzésének leírásában szerepel, hogy egy-egy programozási módszert melyik nyelven fogják megismerni a diákok. Funkcionális: ML, procedurális Java, logikai: Prolog; emellett tanulnak C, C++ nyelvet is. Ez az egyetem választása, az ott képzett programozók ezekről a nyelvekről fognak majd önállóan továbbtanulni, átállni más nyelvekre. Negyedszázada végeztem a számítástechnikatanári szakot, amin a Pascal nyelv mellett tanultam Basic, ELAN, LCN Logo, Assembly, Fortran, Prolog, SQL (ezek nevére emlékszem) nyelveken programot írni. Ezek közül ma az akkor összesen 4 órában tanult SQL-t tanítom alapszinten. Tanfolyamokon, 30 órás továbbképzésen, saját erőből tanultam C#, Java, C++, C, PHP, Visual Basic, Ruby on Rails, Imagine, Scratch nyelven programot írni, de olvasom (megértem) a Python, Java Script, HTML5 és blokknyelveken írt programokat is. Érettségi dolgozat értékelését minden választható programozási nyelven vállalok.

[82] https://www.pagony.hu/mindent-bele-torta-avagy-egy-ragacsos-ulinap [2021.10.30]

Az informatikusnak – azaz programozónak, programtervezőnek, mérnökinformatikusnak, gazdasági informatikusnak, … informatikatanárnak – képesnek kell lennie arra, hogy a munkájához szükséges programozási nyelvet (géppel való kommunikáció nyelvét) megtanulja. Ehhez adnak alapot azok a nyelvek, amelyeket a képzése során tanult meg. Én az ELTE-n sokféle nyelv használatát tanultam, erre alapozva tanultam más nyelvek alapvető eszközeinek a használatát. A szükséges mértékig megtanultam az eszköz, a programozási nyelv használatát.

Példa egy nyelv felhasználói megismerésére:

> A Pascalról át kellett tanulnom egy másik nyelvre, én a C#-ot választottam. Az átálláshoz az addig megoldott feladatok C# nyelven történő megírását választottam. A vezérlési struktúrák könnyen mentek, mert a számlálós ciklus helyett a for-ciklust használtam. Idegenkedtem attól, hogy a for-ciklust hiányosan vagy összetett feltétellel adjam meg. Bár megértettem, soha nem írtam így. Emlékszem, a fájlműveletek (fájl megnyitása írásra, olvasásra) nagyon nehezen ment. A minták alapján a „using System.IO”-t két helyen lehet beírni, a fájl olvasásához elég a Stream.Reader, de más mintákban a File.Reader-t is használják… Vajon melyik a jobb, szebb módszer? Meg tudom tanulni mindegyiket (vagy inkább bármelyiket), de mennyiben mások ezek a megoldások, melyiket tanítsam? Levelezőlistán próbáltam megtudni, hogy kinek mi a véleménye, mi a szokás… De akkor nem kaptam választ.

A nyelv használatának megtanulása olyan, mint egy szövegszerkesztő használatának a megtanulása. Informatikából nem elég a szövegszerkesztő használatának az ismerete, mert a különböző szövegszerkesztők működése eltérő lehet. Tudni kell, hogy a szövegszerkesztő működése mögött milyen programozói döntések vannak, mit tekint alapértelmezettnek, mi az objektum tulajdonsága és mi lesz a függvénye (például bekezdés jobb behúzása és szélessége); mi lesz az eredménye ugyanarra az objektumra ható többféle beállításnak (például stíluslapok esetén).

Egy programozási nyelvet is úgy kellene ismerni, mint egy szoftvert, nem csak a használatát, hanem azt is, hogy a használt eszközök mögött mi fog történni, milyen programozói döntések érvényesülnek, vagy legalább azt, hogy annak hatására valamilyen szabályok alapján fog működni a programom, amit nem tudok befolyásolni. Negyed századon át használtam úgy a string típust, hogy nem tudtam, valójában hogyan kezeli a gép a szöveget. A Pascal – valószínűleg – struktúraként, egybájton tárolt méret majd ennek megfelelően lefoglalt karaktertömb formában kezelte. C-ben viszont lezárónulla jelzi a végét (ami szintén egy bájt), C#-ban egy osztály, amibe nem látunk bele, de ismernünk kell a tulajdonságait, például azt, hogy nem lehet indexszel mutatott karaktert módosítani (immutable). A C++ stringje viszont mutable.

Példa folytatása – felhasználás helyett megismerés

Pascalban tanultam OOP-t (elvileg): Pont, kör, téglalap, háromszög, sokszög. Az, hogy a C# OOP alapú tudtam, de az érettségihez nem kell osztályt létrehozni, eszembe sem jutott, hogy ne típusként kezeljem az osztályokat. Megtanultam, hogy C#-ban itt-ott kell a new és nem értettem, hogy a StreamReader miért van 150 oldallal később a jegyzetben, mint a Consol.ReadLine. A fájlmegnyitási kérdést azzal zártam, hogy a legkevesebb karakter beírását igénylő megoldást választottam – tiszta felhasználói gondolkodás.

A C# nyelven az OOP alapú programozást úgy tanítom, hogy a Windows Form alkalmazásokban Button, Image és egyéb objektumok felhasználásával írunk programot. A blokknyelvek használata is hasonló módon tanítja az OOP paradigmát, felhasználói szemlélettel.

Idővel elkezdtem tanítani osztályok definiálását, mert néhány érettségi feladatban szebb volt az osztályhoz írni a függvényt, mint a Program-hoz és az osztályt szebbnek tartottam, mint a struct-ot. A konstruktorról és destruktorról tudtam, hogy a nyelv elintézi. Majd a mérnökinformatikus képzés kutatása során elkezdtem – a módszertani megfigyelés közben mellékesen – C++-t tanulni, és láttam, hogy C++-ban a konstruktort, konstruktorokat hogyan kell megírni, meg kell írni. Elég sokáig tartott, mire rájöttem, hogy C#-ban mire jó a konstruktor írása, hogy ebben a nyelvben is van értelme többféle konstruktort írni, nem az alapértelmezettet használni. Ekkor értettem meg a StreamReaderről szóló kérdésem lényegét.

Nem tudom, kell-e hozzá kutatás, vagy elegendő a tapasztalatom: az ELTE-n a programozási nyelv egy felhasználói eszköz, a nyelv megismerése azt jelenti, hogy a használatát ismeri meg a programtervező; ugyanígy, felhasználói aspektusból ismeri meg a programozási nyelveket az informatikatanár is. Ezzel szemben a BME VIK-en a nyelv termék is, a programozási nyelv tanulásakor a kód hatását vizsgálja. Ez a szemléletbéli különbség a problémák megközelítési módjából adódik, azonban a később megtanulandó programozási nyelvekhez, azok minősítéséhez, a helyes használat megtanulásához – ahogy az alkalmazói programoknál is láttuk – nem elég a „hogyan használom”, a felhasználói szemléletű ismeret. Szükséges a „hogyan működik” vizsgálata is, az informatikai szemléletű megismerése az adott programozási nyelvnek.

Az ELTE programtervező informatikusai első félévben nagyon sok nyelvet tanulnak meg használni, de egyik nyelvet sem ismerik meg mélyebben. Lehetne a Programozás tárgyban C-t használni és az Imperatív programozás tárgyban megtanulni, hogy hogyan működik a C, hogyan lesz belőle program. Vagy lehetne C++-t tanítani az Imperatív programozás tárgyban is,

bár akkor nehezebb megmutatni az indirekciókat: pointert, referenciát, ami, ha Pythonnal kezd ismerkedni a hallgató, abban még inkább el van rejtve. Bármelyik nyelvet választjuk, a felhasználás mellett szükséges ugyanannak a nyelvnek a programozói megfontolása, a nyelvi paradigmák megismerése, mert ez adja meg az alapot a többi nyelv értelmezéséhez, a helyes felhasználásához.

XIII/2. Párhuzamosan használt nyelvek

Robert C. Martin [157 p:41] programozóknak szóló tanácsa, hogy munka előtt KATA-t kell írni, amely az aznapi munkához szükséges tudáscsomagot (nyelvet, algoritmizálási rutint…) előhozza. Ahogy a sportban az izmokat, a programozásban az agyat be kell melegíteni a munkához. „Elő kell venni” a megfelelő szótárt. Hasonlóan, angoltanár kollégáim a vizsga előtt angol nyelvű kommunikációt, olvasást javasolnak, hogy az angol nyelv szavai, kifejezései előjöjjenek, a gondolkodás átválthasson angolra.

Speciális nyelvi képesség szükséges ahhoz, hogy különböző nyelveket gyakran váltva se keverjünk. Ha két nyelven kell felváltva tanítaniuk, az megnehezíti a kódolást. Megfigyelhető David Malan előadásain is, hogy C-ben „otthon van”, a Pythont csak ismeri.

Példák programozási nyelvek párhuzamos használatára

A CS50 2018 – Lecture 2 – Arrays[83] előadás 16. percétől Malan magyarázat közben egy ciklust ír, néha belenéz a jegyzetébe, de a kódon nem kell javítania. Később rájön, hogy előbb mást szeretne bemutatni. Beszéd közben módosít, futtat… A Python bevezetésénél: Python – Intro to Computer Science – Harvard's CS50 (2018) – Lecture 6[84] sokkal többször néz a jegyzetbe, többször rontja el a gépelést (érdemes megnézni például 1:00:00-tól). Jól látható, hogy mindkét nyelvet ismeri, de a keze a C-re áll rá, a Python esetén előadás közben még azon is kell gondolkodnia, hogy hogyan írja pontosan.

A *Programozás alapjai 1*. előadás arról (is) híres, hogy az előadó úgy írja a C nyelvű kódot, hogy közben magyarul mondja a tartalmát. 2018 őszén párhuzamosan adott elő C és Python nyelven, az egyik Pythonos előadásán a C prinf utasítását írta le a print helyett. Egyszer… és ezt a hallgatók Facebook bejegyzésben megjegyezték, mert kuriózum volt. A gyakorlatokon kiderült, hogy az előadónak nem okoz gondot a C és C++ elkülönítése,

[83] https://www.youtube.com/watch?v=u-kH-5JJSgU [2021.10.30]
[84] https://www.youtube.com/watch?v=hnDU1G9hWqU [2021.10.30]

képes rugalmasan váltani közöttük, a Pythonra váltás is csak pillanatnyi megakadást jelentett – miközben a mindennapi munkája során még további nyelveket is valószínűleg „folyékonyan” használ.

A saját tapasztalatom ennek pont az ellenkezője. 2018 őszén keddenként C#-ban tanítottam programozni a Szent István Gimnáziumban (SZIG), szerdán C++-ban írtam programot az ELTE-n, csütörtökön C++-ban érettségi felkészítést tartottam a Veres Péter Gimnáziumban (VPG). Az eltérő környezet sem segített. A VPG-ben nem emlékeztem, hogyan kell fájlból beolvasni adatokat, a példányosításnál mindig elakadtam azon, hogy kell-e „new”. A List<int>-et és a vector<int>-et folyamatosan kevertem – mindkét gimnáziumban mindkettőt használtam, az ELTE-n elkerültem.

Az elakadásaimmal együtt is, 2018-ra már sokat fejlődtem az egy évvel korábbi állapotomhoz képest. 2017. februárban egy C#-ot tanító időszakomban, az első C11 és C++11 gyakorlaton a három nyelv még teljesen kiütötte egymást: Egy feladat megoldásához negyedórán keresztül gondolkodtam, hogy a „struct” vagy „class” szóval kell-e kezdeni. Az előző feladatot C-ben kellett írni, váltani kellett C++-ra, de az miben is különbözik a C#-tól… Végül a neten C++ kódokat olvasgattam, a feladattal nem haladtam semmit – és kétségbeesve saját állapotomtól jöttem el a gyakorlatról.

Ezek a példák nekem azt mutatják, hogy különböző nyelveken programozni lehet, de mindenképp megterhelő. Van, akinek a plusz terhelés „nem számít”, másokat megakadályoz a gondolkodásban, ami kisebb-nagyobb mértékben rontja a munka hatékonyságát.

Az ELTE IK első félévében tanított nyelvek sokféleségének oka – úgy tűnik – éppen ez a plusz terhelés. Nem szándékos nehezítés a hallgatók számára, hanem korlát az oktatók részéről. A „matematikusok” Pythont kértek, mert azt tudják ők (a C++ nehéz), a Programozás tárgyon C++ a legoptimálisabb, mert arra vannak oktatók… és mindegyiket első félévben kellene tanítani, mert a következő félévekben már használtatni szeretnénk.

7.3.6

XIV. Az informatika (programozás) tanítása

XIV/1. Integrált „Guess the Code”

Az egyes alkalmazások funkcióinak tanításakor elő lehet készíteni a programozás fogalmait. Vagy inkább: informatikát és programozást tanítunk akkor, amikor egyes alkalmazások funkcióit vizsgáljuk. Ezt egészítik ki olyan „Guess the Code” feladatok, amelyekkel a szoftver belső működésével kapcsolatosan fogalmazunk meg elméleteket.

Példák „hogyan működik” feladatra

- Linkeskedés

Mi a különbség egy csatolt (belinkelt) és egy beillesztett objektum között? Egy képet Wordbe, Excelbe, PowerPointba, Prezibe illetve weblapra szeretnénk betenni. Melyik módszer alkalmazható az egyes esetekben?

- Szöveg egységek

Rendezd tartalmazási sorrendekbe a szöveg egységeit: bekezdés, dokumentum, karakter, mondat, oldal, sor, szakasz, szó.

- Alatta-felette

Egy bekezdés sorainak távolságát a sorköz mértéke határozza meg. A bekezdések közötti távolságot pedig a bekezdés előtti és utáni térköz. A beállított sorköz hol jelenik meg a sorhoz képest? Hogyan módosítja a sorköz a beállított térköz megjelenését? Milyen távolság lesz két bekezdés között, ha mindkettőre be van állítva valamekkora sorköz ($s1$, $s2$) és az első bekezdés után ($tu1$) továbbá a második bekezdés elé ($te2$) különböző mértékű térközök is be vannak állítva? Add meg a $táv(s1, s2, tu1, te2)$ képletét!

- Négyzetkilométer

Hogyan lehet Wordben, PowerPointban megjeleníteni a km^2 2-es számjegyét? Hogyan jeleníthető meg ugyanez Excel cellában, ha szövegként akarjuk beírni? Mi a megoldás akkor, ha formátumként szeretnénk megjeleníteni a kitevőt? Van, amikor nem a kilométer négyzetét kell megadni, hanem egy számét, pl. 3^2. Ezt a jelölést hogyan érhetjük el az Excel egy cellájában?

- Mennyi az annyi?

Az osztályátlagnál a századok is számítanak. A számítást azonban többféleképpen lehet végezni. Egyik módszer az összes diák összes jegyéből számított átlag. Másik módszer, hogy a diákok átlagából számítjuk az osztályátlagot. Lehetne a tantárgyak átlagából

is osztályátlagot számolni. Saját adatok vagy a mellékelt Osztálystatisztika.xlsx adatai segítségével mutass olyan – valóságban is lehetséges – esetet, amikor a három átlagszámítási módot két tizedesjegy pontossággal alkalmazva eltérő eredményt kapunk! (Üres és szöveges cellák miatt lesz eltérés.)

A „hogyan csinálja” kérdésen túllépve, arról is érdemes beszélni, hogy miért így vagy úgy csinál valamit egy szoftver. Egyes esetekben az alternatív megoldások a szoftverek területén is alternatívákat jelentenek, eldönthetjük, hogy nekünk melyik szimpatikusabb.

Példák „melyik jobb” feladatra

- Képecske

  Adott egy fénykép, aminek azonban csak egy részét kellene egy dokumentumban (vagy prezentációban) megjeleníteni. Milyen módszerekkel oldható meg a feladat? Mi az egyes módszerek előnye, illetve hátránya?

- Jelenléti ív

  A jelenléti ívek egy hétre szólnak, de van, amikor egyes napokon nem kell vezetni. Ezt az íven jelölni kellene. Az alábbi megoldások körül melyik a gyorsabban kivitelezhető? Melyik a használhatóbb?

1.

Monogram	Hétfő	Kedd	Szerda	Csütörtök	Péntek
BP					
Fe					
Szl					

2.

Monogram	Hétfő	Kedd	Szerda	Csütörtök	Péntek
BP	~~~~~~~~~~~~	~~~~~~~~~~~~	~~~~~~~~~~~~		
Fe	~~~~~~~~~~~~	~~~~~~~~~~~~	~~~~~~~~~~~~		
Szl	~~~~~~~~~~~~	~~~~~~~~~~~~	~~~~~~~~~~~~		

3.

Monogram	Hétfő	Kedd	Szerda	Csütörtök	Péntek
BP	BPBPBPBPBP	BPBPBPBPBP	BPBPBPBPBP		
Fe	FeFeFeFeFeFe	FeFeFeFeFeFe	FeFeFeFeFeFe		
Szl	SzlSzlSzlSzlSzl	SzlSzlSzlSzlSzl	SzlSzlSzlSzlSzl		

Vannak olyan alternatív lehetőségek is, amelyek csak első pillanatra tűnhetnek jobb – például a felhasználó számára kényelmesebb – megoldásnak.

Példák „másképp is működhetne” feladatra

- Szóközök – a végtelen rekurzív csere

Gyakori eset, hogy egy dokumentumot át kell szerkeszteni, de rengeteg felesleges szóköz van benne. A javítás legegyszerűbb módja, ha két szóközt egy szóközre cserélünk. Így 10 egymást követő szóközből 5 lesz… Egy dolgozatban az első sor behúzása helyett 7 szóköz szerepelt, az egyes fejezetek között 37 szóköz után 3 csillag majd újabb 23 szóköz volt. A két szóközönkénti cserét hányszor kell végrehajtani, hogy mindenhol csak 1 szóköz legyen? Ismerve az egymás utáni szóközök számát, hogyan lehetne optimalizálni a cseréléseket?

Milyen csere algoritmus kellene ahhoz, hogy a cserélendő 2 szóközt 1 szóközzel helyettesítő műveletet egyszer végig csinálva a teljes szövegen az összes felesleges szóköz eltűnjön? (Másképp: azt szeretnénk, hogy ismétlődő karakterek tetszőlegesen hosszú sorozatából csak egyet hagyjon meg.) Ez a speciális csere nagyon meggyorsítaná a munkát számos esetben. Miért nem így írták meg a Wordben?

- Óraterv – formátum felismerése és félreismerése

Előfordulhat, hogy percekre lebontott óratervet/műsortervet kell készíteni. Az egyes tanórai tevékenységekhez rendelt időt többféle formában megadhatjuk. Ugyanaz a „10 perc” lehet 10, 10 perc, 0:10, 0.10, 10 p, 10 min. Melyek a lehetséges megjelenítési formák közül az értelmesek? Mi a hiba a nem értelmezhető megadási formákkal? Melyek azok a formák, amelyek ugyan hibásak, de függvénnyel könnyen értelmessé tehetők? (Értelmesnek tekinthető minden olyan megjelenítési mód, ami lehetővé teszi a tanóra 45 perces hosszának ellenőrzését SZUM() függvény segítségével.)

Nem csak az új ismeretet adó órák, hanem a gyakorlófeladatok is lehetnek többcélúak. A kiadott feladattal a tananyag megfelelő részét gyakorolják a diákok. Emellett a téma megválasztása támogathatja más tantárgyak tanulását, motivációs tényező lehet. A feladat kiadásának a módja tartalmazhat informatikai, programozási többlet ismeretet.

Példa arra, hogy a feladat kiadásának módja jelentősen megváltoztatja a feladat funkcióját:

Az eredeti feladat: rajzolj magyar zászlót Paint programmal 600×400 pixeles méretben. A módosított feladat: rajzolj minimális számú kattintással magyar zászlót Paint programmal 600×400 pixeles méretben.

A módosított feladat játékosabb, izgalmasabb, emellett a megoldás tervezését – algoritmustervezést – igényli. Az algoritmust le lehet íratni, el lehet mondani. A Paint eszközeinek alaposabb megismerését is motiválja, e mellett egy hatékonyságra törekvő algoritmizálási feladatot is meg kell megoldani.

8.1.2

XIV/2. Mini-projekt

A „tipikus”, életszerű problémák a legegyszerűbb feladatot is kreativitást igénylő feladattá tehetik. Például az alkalmazói versenyek jellemző „kihívása” az adatok használható formátumúra alakítása: a forrásból másolás során rendezetlenül kapott adathalmazból strukturált, táblázatos elrendezés kialakítása, konverziós problémák kezelése.

A projektek jellemzője, hogy a téma határozza meg a feladatot, ehhez több területről kell felhasználni ismeretet. Az informatika oktatásában nagyon jó hatása van a projekteknek, az oktatás során elsősorban az egyéni projekteknek. A különböző kérdőívek adatainak feldolgozása könnyen elvisz az adatok és adattípusok értelmezése irányába [28]. A feladatgyűjteményekben nagyon sokszor találunk placebo – például fiktív esemény, egy tanulócsoport esetén nem létező név – adatokkal feladatot. Ezeket a feladatokat a helyi viszonyokra, az adott tanított csoportra vonatkozóan át lehet fogalmazni, ami nem csak érdekesebbé teheti a feladatot, hanem további informatikai témák tanulását is eredményezheti.

Például egy, a csoporttagok születésnapjaival kapcsolatos feladat a táblázatkezelés, dátum-idő kezelése és függvényei mellett a közös munka megszervezésére is feladat – hogyan gyűjtsük gyorsan össze egy táblázatba a születésnapokat, hogyan kapja meg mindenki az összes adatot. Továbbá: kötelező-e igazat mondani, kinek lehet kiadni az összegyűjtött adatokat? Más kihívást ad az elektronikus naplóból táblázatba átvenni osztályzatokat.

Egy programozás feladat több szempontból is projektnek tekinthető. „Készíts programot…” feladat megoldása a projekttevékenység összes részfeladatát tartalmazza, legfeljebb a dokumentáció elkészítése hiányzik.

8.1.2
8.1.4

XIV/3. Elnyújtott, átfedő LAU-ok

Egy-egy LAU tanítása hónapokon, éveken át tarthat. Az egyetemi képzésben a programozás heti 4–6 kontaktóra. Lehet, hogy ez egy kicsit sűrű, de a közoktatásban jellemző heti 1 óra nagyon rossz hatékonyságú. Például a programozás kezdeti szakaszában valószínűleg az ideális heti 2-szer 2 óra lenne, de általánosan is, ahhoz, hogy egy-egy témával haladni lehessen, hogy

a tanórák közötti szünet (LAU^L4) ne legyen túl hosszú, ahhoz a heti 2 alkalom lenne megfelelő. Akár a nyelvtanulást nézzük, akár a testedzést, egyiket sem lehet hatékonyan fejleszteni heti 1 alkalommal.

Programozás esetében több éven át kísérleteztem a heti 1 órás oktatással, körülbelül a 8. héten megakadt a tanulás. Ennyi idő alatt annyi új ismeret tanulása, gyakorlása halmozódott fel LAU^L1, LAU^L3 szinten, hogy a tanóra kevés volt az összes LAU gyakorlására. Például, az első alkalommal megismert string típus három órán szerepelt, de utána egy hónapon át nem volt róla szó. Ha a string fogalma a programozás kezdetén lép be, akkor a 8. órán olyan, mintha nem is hallottak volna róla korábban. Azzal, hogy a digitális írástudás minden lehetséges pontján programozást is tanítok, akár 3–4 éven keresztül készítem elő a programozás tanítását, a string fogalmát is. Lényegében informális ismeretként, mint a médiában a reklám, rögzülnek a diákok fejében a fogalmak. Ilyenkor a stringre a 8. órán elég pár percben visszatérni, jobbára csak a jelölést kell ismételni.

Ugyanezt fordított irányban is alkalmazni kell. Egy már tanított témára a későbbiekben érdemes hivatkozni. Hatékony megoldásnak tűnik erre az azonos feladat többféle megoldása. A szövegszerkesztésben tanultak alkalmazása – és a különbségek megfigyelésére is jó, ha weblap és prezentáció-készítésekor ugyanaz a téma, esetleg ugyanaz a szöveg szerepel. Táblázatkezelés, adatbáziskezelés és programozás esetén az alkalmazói ismeretek ismétlését is jelenti az azonos feladat. A körlevélkészítést az adatbáziskezelés jelentéskészítésével érdemes ismételni (vagy fordítva).

Egyes műveletek, gondolatmenetek ismétlése is hasonlóan beépíthető. Például a „csere” műveletet minden alkalmazásban, valamint a programozásban is érdemes megemlíteni. Mindig egyre rövidebb időt rászánva. A lineáris keresést már a szövegszerkesztés elején és a fájlkezelés tanításakor lehet tanítani; ha robot programozása is lehetséges, akkor a fal keresése is ez az algoritmus; azután 2-3 alkalommal táblázatkezelés feladatokhoz kapcsolva ismételni, majd programozás tanulásakor kódoljuk az algoritmust. Bár az ember jellemzően szkennelve keres, ha a lehetséges pontokon megbeszéljük, hogy a gép hogyan végzi a keresést, akkor ez lesz a természetes a programozás tanulásakor. A dokumentumokban egy szóra rákeresés során a keresés megfigyelése körülbelül 10 másodperc. Mégis elülteti a gondolatszikrát a diák agyában.

8.1.4

XIV/4. Másképp tanulók

> *„A diszlexia viszonyfogalom, diszharmónia a gyermekkel szembeni elvárások, az olvasás-írás tanulására szánt idő, valamint az eredmény között… A tanulási zavarokkal küzdők azért vannak hátrányban az iskolában, mert az oktatás nem illeszkedik képességeikhez. Képesek megtanulni írni, olvasni, számolni, csak nem úgy, mint a többiek. Ezért kell inkább „másképp tanulóknak", mint tanulási zavarokkal küzdőknek nevezni őket."*[85]

Az idézet nemcsak a másképp olvasókra érvényes. A tanulási zavar csak az oktatásban alkalmazott módszerekhez képest értelmezhető, a tanuló agya másként működik, mint ahogy azt az „oktató rendszer" elvárja. Az oktatási rendszer rugalmatlanságának a következménye a felmentés. Az igazi megoldás a másként teljesítés vagy hosszabb idő alatt teljesítés lenne. A másképp tanulók képzésére informatikából is meg kell találni a megfelelő módszereket. Kutatások, majd erre alapozva a tanárok, tanulók és szülők számára módszertani ajánlások kellenek.

Programozási nyelvek tanulása diszlexiásoknak

Példa a diszlexia kompenzálására

> Ma már tudom, hogy mit csináljak, ha „leblokkolok" kódolás közben: gyorsan előveszek egy, az adott nyelven írt mintát vagy elmondom, hogy nem jut eszembe csak a másik nyelven. Nagyon valószínű, hogy diszlexiás vagyok, de nincs róla papírom – az én gyerekkoromban nem volt szokás a vizsgálat, akit vizsgáltak annak csak rosszabb lett, mert értelmi fogyatékosnak tekintették. Nem csak programozási nyelveknél jelentkezik a kommunikációs probléma, hanem idegennyelvek használatánál is. A német nyelvvizsgámon a szövegértés olvasva 85%, hallásután 80%, szövegalkotás írásban 53%, szóban 40%. Írásban szavanként ellenőriztem, hogy németül írtam-e, de a párbeszéd egy részét két nyelven mondtam: szavanként angolul és németül. Mindennapi élethelyzetben percekig nem tudom eldönteni, hogy angol vagy német szöveget hallok-e, és angolul csak akkor tudok beszélni, ha előtte két napig csak angolul kommunikálnak velem. Így Linzben nem tudtam angolul, de Balin igen.

A diszlexia nem betegség, hanem másképp gondolkodás. A diszlexiás nehezebben tanul olvasás útján, de ezt képes kompenzálni más tanulási módszerekkel vizuális vagy hallás útján. Eszerint a diszlexiásnak fontosabb az előadáson részvétel, a beszélgetés; az algoritmizálásban

[85] http://www.diszlexia.info/dislzavaregyform.htm [2021.10.30]

többet segíthet a blokknyelv, a struktogram. A változónevekből valószínűleg inkább a rövidebbeket szereti, mert a hosszú, hasonló változónevek (például csak egy ’s’-ben eltérő) megkülönböztetése nehéz; könnyebb megjegyeznie, hogy mit jelent az az egy-két karakter. A diszlexiásoknak a közoktatásban sokkal nagyobb energiabefektetés szükséges a nyelvek, a könyvből tanulandó tárgyak esetén, amit logikus gondolkodási módszerekkel tudnak kompenzálni, ezért valószínűbb, hogy matematikából jobbak. Ismerőseim ezt igazolják, de tudományos felmérést nem találtam erről.

Azt mondják, a diszlexiások nem tudnak megtanulni nyelveket, fel kell menteni őket a többnyelvűség alól. Környezetemben azt tapasztalom, hogy ez részben igaz. A nyelvek párhuzamos tanulása jelentősen megnehezíti a nyelvtanulást. Például a közoktatásban erőltetett tanterv és nyelvoktatási módszerek akár lehetetlenné is tehetik a követelmények teljesítését. Egymást követően tanulva a nyelveket, meg lehet tanulni több nyelvet is, de együtt használni még ekkor is nagyon nehéz. Ahogy egy kolléganő mondta: persze, mert a különböző nyelvek szavai szinonimák. Éveken keresztül rendszeresen használva mindkét (vagy három) nyelvet, kifejezéseket megtanulva lehet szétválasztani a „szótárakat”.

Példa a diszlexiás hátrányára programozásból

> Programozási nyelv esetén a tanulás nagyon hasonló az élő nyelvhez. A C# és C++ együtt használata számomra egy C++ gyakorlattal kezdődött, ahol elakadtam az első, „class” szónál és gyakorlat végéig nem is írtam semmit. Minden blokkolásra emlékszem, mert nagyon kellemetlen élmény, de a Pascal és C# párosítás nem volt ennyire drasztikus. Ha még 1-2 évig gyakorlom a párhuzamos tanítását ennek a két nyelvnek, akkor lehet, hogy blokkolás nélkül megúszom a tanévet. Csak egy gond van: élő nyelvből 2, esetleg 3 ismerete elegendő, de programozási nyelvből ennél jóval többet kell megtanulni. Remélem, két nyelvvizsga elég lesz. Tudom, hogy ha úgy adódik, hogy nem az anyanyelvemet kell napi szinten használnom, azt meg fogom tudni tanulni. Nem könyvből mondvacsinált témaköröket, hanem kommunikációval valódi élethelyzeteken keresztül. A programozási nyelvektől jobban tartok. Úgy tűnik, évente kellene új nyelvet tanulnom, de nem gyakorlati feladat miatt, nem használat során egy program megírása érdekében, hanem „könyvből”, azért, hogy tanítsam valakiknek.

Az informatika oktatása módszertani kutatásainak nagy figyelmet kellene fordítania a tanulók eltérő képességeire, a képességekre jellemző hatékony tanulási és tanítási módszerek meghatározására. A diszlexia módszertani megfontolásokat igényel, nem felmentést. A diszlexiásnak az a segítség, ha kevés lexikális ismeretből kreativitással tud sokat kihozni.

Diszgráfia kompenzációja számítógéppel

A diszgráfia sokkal látványosabb képességeltérés, pusztán azért, mert a csúnya, olvashatatlan írás vagy a durva helyesírási hibák feltűnőek. A diszgráfia is tanulási nehézség, azaz nagyon sok gyakorlással javítható az íráskép, ha a képességfejlesztés szakszerű és elég időt adunk rá. A számítógépes kommunikáció a kézírás alternatív megoldása, ezért valószínűleg az átlagnál több diszgráfiás választja az informatikát. Emellett a diszgráfiára hasznos felmentést kérni a kézírás alól, helyette gépen teljesíteni a követelményeket.

A diszgráfiának nem csak a csúnya írás a jele, tanulási, illetve teljesítménybeli problémát okozhat a bal-jobb orientáció bizonytalansága. Programozás esetén ez különösen a relációjelek tévesztéseként jelent gondot, mert a betűcserét, kihagyást a fejlesztőkörnyezet támogatásával jól lehet kompenzálni. A relációjelek helyes használatát – tapasztalatom szerint – úgy lehet biztosítani, hogy mindig a számegyenesen elhelyezés sorrendjében adjuk meg a mennyiségeket és mindig a '<' karaktert használjuk. Akik nem diszgráfiások, azok jellemzően az „úgy írom, ahogy mondom" elvet használják, hiszen megtanulták – és pontosan tudják – hogy melyik a kisebb és melyik a nagyobb jel. Van, akinek a rendezettségi sorrend sem elég, mert nem „látják, hogy a számegyenesen merre van a nyíl", nekik a teszt jelent segítséget. Azonban bármi legyen a megoldás, valószínűleg hosszabb ideig tart a megvalósítás, mint a „normálisaknak".

A diszgráfia informatikából megfelelő módszerekkel kompenzálható, de a felismerést és hatékony módszertani eszközök bevetésének módját még kutatni kellene. Például C++ esetén az inserter és extractor hibás használata előre jelezheti, hogy a szélsőérték-keresés és a rendezés algoritmusok során gond lesz. De ez csak akkor valósul meg, ha a C++ az első nyelv, amin ezeket az algoritmusokat tanulja a diák. Egyéb esetekben jelzés értékű a relációsjel tévesztése, amit csak akkor veszünk észre, ha figyeljük a diák tevékenységét (vagy látjuk első kísérleteit). Ilyenkor érdemes „elbeszélgetni" a diákkal, meg kell kérdezni, hogy hogyan gondolkodik a jelekről és számára megfelelő gondolkodási sémát kell javasolni – ami nem az „úgy írom, ahogy mondom", mert láthatóan ez nem megy.

A diszgráfiásnak a relációkkal később is lehet gondja. A komparátor függvények –1, vagy 1 eredménye azon múlik, hogy a bal oldalon vagy az elől lévő érték a kisebb-e. Aki bizonytalan abban, hogy melyik a bal oldal, melyik van elöl (gondoljunk arra, hogy a diszgráfiás lehet, vagy balkezes és tud bal kézzel jobbról balra is írni, de legalábbis úgy érzi, neki mindegy) annak ezek a megfogalmazások nehézséget jelentenek.

Azt tapasztaltam, hogy a diszgráfia nem akadályozza meg az informatika tanulását, alkalmazását, de bizonyos műveleteknél a diszgráfiás megakad, mert gondolkodnia kell olyan dolgokon, ami másoknak „zsigerből megy”, eközben a könnyebben elfelejti az eredeti feladatot.

Diszkalkuliás informatikus

A közoktatásban ismert jelenség a diszkalkulia, ami a matematika tanulmányokat akadályozza; számolási, mennyiségek értelmezésével kapcsolatos gát. Informatikatanárok beszélgetései alapján úgy tűnik, sok diszkalkuliás, aki felmentést kap matematikából, az informatikát választja vizsgatárgynak. Azonban ezek a tanárok semmit sem tudnak arról, hogy a diszkalkuliásoknak milyen módszerekkel lehet jól informatikát tanítani, tudjuk-e informatikai módszerekkel kompenzálni a matematikai gátlásokat. Azt sem tudjuk, mi az, ami miatt informatikából biztosnak látszik az elégséges azok számára, akik matematikából megbuknának. Reményre ad okot, hogy több eset ismert, amelyben matematikából gyenge tanuló programozói szakot végzett, ugyanakkor nem elhanyagolható, hogy a matematika eredmények – a jelenlegi módszerekkel tanítva – erősen korrelálnak a programozási eredménnyel.

Azt gondolom, hogy több diszkalkuliás diákom volt, közülük kettőnek „papírja” is volt erről. A számokkal való „rossz viszony” egyiküknél sem jelentett akadályt a logikus gondolkodás területén. Az egyik vizuális érdeklődésű volt, fényképezéssel, grafikával, webtervezéssel és mára programozással foglalkozik. A másikat csak középiskolás korában figyeltem, ő szöveges adatokkal elég jól boldogult, még táblázatkezelésből is. Szintén a konkrétan látható dolgokkal tudott foglalkozni, az elvont, „lelki szemeim előtt megjelenő” dolgokkal nem. Például nem „látta” az átlagot, de szöveges, kommunikálós program írása nem jelentett nehézséget számára.

Az egyetemi tömegképzés velejárója lehet, hogy főleg mérnökinformatikán megjelennek enyhén diszkalkuliások. Lehet, hogy emiatt érzik az oktatók, hogy a „mai hallgatók nem tudnak számolni”. A diszkalkuliáról szóló leírások kiemelik, hogy ez a képességzavar nincs összefüggésben az intelligenciával, ezért a számolási nehézségek – lassú műveletvégzés, csak írásban vagy géppel képes összeadni, leblokkol számolási utasításra, nem veszi észre, hogy nagyságrendileg irreleváns az eredmény – kompenzálására megfelelő módszereket kell keresni. Az egyik módszer épp a számítógép használata lehet.

Asperger-szindróma

A magasan funkcionáló autizmust nevezik Asperger-szindrómának (röviden aspie). Ez azt jelenti, hogy az Asperger-szindrómás a környezetével kommunikál, de társas kapcsolatokkal gondja van. Jellemző, hogy inkább géppel kommunikál, mert a gép érzelmeivel nem kell számolnia. A magas IQ-val rendelkezők kiváló programozók, informatikusok lehetnek. Jellemző,

hogy az emberek megismerése helyett inkább a gépek, a fizika megismerése köti le őket, csak ezzel foglalkoznak, azaz „kockák”.

Asperger-szindrómát csak az utóbbi években diagnosztizálnak. Az első diagnózisról úgy szereztem tudomást, hogy előtte az illetőt már 4 éve ismertem. Amikor elmondták, hogy ez a diagnózis miről szól, rögtön megneveztem másik diákot is, kiderült, hogy neki is van „papírja”. Informatikatanárként fontos tudni, hogy az aspiesok nehezen ismerik ki magukat az emberekkel való kapcsolatokban, annál inkább jó „társ” számukra a gép. Informatika szakköret kedvelik, mert ott azzal foglalkozhatnak, ami leginkább érdekli őket, az élményeiket az ott levő informatikatanárral osztják meg, vitatják meg.

Tanárként fontos tudni, hogy az ilyen diákok a társas kapcsolatokat tanult szabályok alapján „intézik”. Sorban állnak akkor is, ha a többiek nem, jelentik a puskázást, egy nap akár tucatszor is köszönnek, minden választ megköszönnek… A szabályokat nagyon komolyan betartják, ha mégsem, akkor büntetés jár nekik; ha másképp nem, akkor önbüntetéssel oldják meg az ügy igazságos elintézését. Elvárják maguktól a teljesítést, nagy lelkesedéssel dolgoznak akár a lehetetlen eléréséért is, a sikertelenség depressziót okozhat. Az „isten vagyok” nagyon aktív, és a „senki vagyok” önpusztító állapotok váltakozását kell a környezetének – elsősorban az általa elfogadott mentorának, tanárának – kiegyensúlyoznia, mindkettőből(!) visszatartania.

Informatika (programozás) egy-egy témájából rendkívül tájékozottak lehetnek, önállóan tanulnak – minden elérhető forrásból –, az adott témában a tökéletességre törekednek, a lehetetlent kívánják elérni – mert a tökéletesség elvárható. A céljaik megvalósítása során jellemzően nem figyelnek a környezetre – mint a tank, törnek a célfelé.

Példák a tökéletesség vágyára:

- Tökéletes kódoló: fejben kigondolja a programot, kódolja – már ellenőriznie sem kell, mert az úgy tökéletes, soha nem téveszt kódolás közben. Képtelen csapatban dolgozni, mert a megoldás a fejében van, nem mondja el, mert az „sok idő”, így nem lehet ellenőrizni, nem lehet megvitatni vele a megoldás részleteit. A csapattagok unatkoznak, amiért mérges rájuk, hiszen csak ő dolgozik a csapatért. Hosszú viták szükségesek arról, hogy tökéletes ember nem létezik így tökéletes kódoló sem, mert az is ember és ő is ember, aki tévedhet.
- Tökéletes grafika: Miért nincs két szomszédos pixel között még pixel? Hiába digitális tárolásról lenne szó, attól még kellene „ott” lennie még egy pixelnek, mert az szükséges neki. Emiatt nem fog sikerülni a kard megjelenítése úgy, ahogy elképzelte.

- Tökéletes LEGO robot: a robotépítő versenyekre a legösszetettebb robotokat mindig az Asperger-szindrómás diákjaim készítették. A robot bonyolultsága miatt az építési idő kevés. A robot mérete a lehető legjobban kitölti a rendelkezésre álló teret. A bonyolult megoldásokban nagy ötletek rejlenek, de mindegyik külön-külön megoldás egy-egy problémára. Szintén Asperger-szindrómások készítették a legjelentősebb LEGO modelleket, vonalkövető autót.

Ugyanez a nagy figyelemkoncentráció a feladatok megoldásában problémát jelenthet. A feladat szövegének egyetlen szava vagy nem pontosan definiált feltétel is félreviheti a feladat értelmét, aminek hatására az egyszerű feladatból nagyon bonyolult problémát – új feladatot – kreálnak maguknak. Bíznak a tanárban (aki a feladatot készítette) és úgy gondolják, hogy csak pár lépés és meglesz a megoldás a helyesnek vélt úton. Mindig csak még egy-két lépés… Eközben kifutnak az időből és rossz esetben teljesen más feladaton dolgoznak, ami értékelhetetlen. Az aspies diákra jellemző, hogy nagyon változó a teljesítménye. Ha ráérez, pár perc alatt megoldja a feladatot, máskor, akár ugyanolyan típusú feladatra értékelhetetlen megoldást ad vagy olyan bonyolultat, hogy komoly kihívás a gondolatmenet ellenőrzése.

Példa feladatmegoldásokra egyazon diáktól:

- Táblázatkezelés dolgozatban még a keresési feladatot is SZUM(HA()) tömbfüggvény használatával oldja meg. Mert ez tetszett neki legjobban. A mindenre jó megoldási módot szó szerint mindenre használta.
- Egy érettségi programozás feladatban a beolvasásig sem jut el. Nagyjából az okozott gondot, hogy nem értette meg a feladatban leírt modellt. Egyenletes mozgás és a minta adatok tárolása helyett „mi van ha” – „mert a valóságban az is lehet” – kérdései voltak. Két héttel később 40 perc alatt megold egy másik feladatot. Közben a történelemtanárát lecserélték, ezért ismét tudott aludni éjszaka.
- Történelemből a legjelentősebb kisebbség megnevezésekor azt a népcsoportot próbálja megadni, amelyik legkisebb létszámú. Rájön, hogy a legkisebb lehetséges érték a 0, de akkor olyan népcsoportot kellene megneveznie, akik nincsenek… A legnagyobb létszámú népcsoportot viszont nem tartja jelentős kisebbségnek, hiszen azok száma nem jelentősen kisebb. Ehhez hasonló értelmezések sokasága miatt felmentik az írásbeli vizsga alól, mivel szóbelin néhány szóval helyes útra lehet terelni a gondolatát.
- BME VIK villamosmérnök 1 félév 1. programozás ZH 1. feladat: A beugró részt elolvassa, ír egy megoldást. A feladat folytatásában használnia kell a beugróban írt

> függvényt, de az ő megoldása nem alkalmas erre, mert az bonyolultabb, kicsit más funkcióra készült. Ezért sok időt veszít, végül egy elképzelt függvényt (ami a helyes lett volna) használ fel a folytatásra. Az eredmény: ez a feladat 0 pontos, mert az eleje hibás, a második rész pedig nem a megírt részt használja fel. Az időt elvitte a feladat, így elégtelen lett a ZH. A javítóra szinte tökéletes megoldásokat ír. A tananyagot már akkor tudta, amikor beiratkozott.

Nem csak a példában említett diákok kaptak részleges felmentést, és akik kaptak, nem ugyanazt kapták. Jellemző, hogy egy-egy témában tehetségesnek mutatkoznak, már amennyiben a tehetségen főleg a kitartó célirányos szorgalmat értjük. Továbbtanulásuk azonban nagyon rizikós. Aki úgy megy egyetemre, hogy a teljes első év anyagát ismeri, annak van fél éve, hogy megtanulja az egyetemi viszonyokat. Továbbá arra felfigyelnek a tanárai – mint nagyon okos hallgató, „zseni” – és a kiemelt figyelem alkalmas arra, hogy a problémákat leküzdjék. Míg a disz-esek konkrét tanulási problémája pontos diagnózissal és erre szabott oktatásmódszertani eszközökkel megoldható (a felmentés nem megoldás), az aspiesoknak kontrollra van szüksége. Olyan személyre, akitől lehet kérdezni (ZH alatt is), akinek elmondhatja, hogy mit gondol éppen, aki szól, ha félreértelmezett valamit. Az aspies egy kicsit robotnak látja magát, szüksége van egy debugerre. Furcsa módon, az aspies a számítógépes gondolkodásban a legjobbak között van, de gondolkodása rugalmatlan, ezért pont az előnye lesz a hátránya is. Zseniális ötletei, megoldásai vannak, de ugyanazzal a lendülettel nagy melléfogásai is.

Tanárok között is van Asperger-szindrómás, de aspies tanítványaim is sokszor tanítottak. Tanárként, oktatóként, korrepetitorként az aspies határozott elképzelésekkel rendelkezik arról, hogy mit kell mondania, amit mond azt a diákok, hallgatók nyilván megértik – hiszen ő is érti. Nagyon (néha túl) részletes, pontos, esetenként érdekes előadást tart, főleg a kedvenc témáiból, de a hallgatók jelzéseire nem tud figyelni. Amikor csak 1-1 emberre kell figyelnie (például korrepetálás), akkor kitűnő tanár lehet, mert „minden” kérdésre tudja a választ, előre látja a tanítvány gondolatában a hibát és pontos indoklással ad útmutatást az általa helyesnek tartott megoldás felé. Néhány dologra kell figyelnie: Mi az, amit a tanítványnak meg kell tanulnia – nehogy valami olyat akarjon tanítani, ami majd csak a PhD-hez – vagy ott sem – szükséges; a tanítvány érti-e a magyarázatot, vagy csak hallja; ne oldja meg a tanítvány helyett a feladatot.

8.2

XIV/5. Nemi szerepek és az informatika

Charles Babbage volt az első programozható számítógép tervezője. Ada (Byron) Lovelace pedig az első, aki programot publikált. Babbage építette (majdnem készre), Lovelace a felhasználás módjáról írt. A magyar (és nemzetközi) informatikatörténet jelentős alakjai férfiak, de ha megnézzük a szegedi informatikatörténeti kiállítást, azt látjuk, hogy az első nagy gépeken sok nő dolgozott. A XXI. században az informatikai eszközöket férfiak és nők egyformán használják. Mégis a programozók, az informatikusok nagyon nagy arányban férfiak. A BME VIK-en és ELTE IK-n vannak olyan tanszékek, ahol csak az adminisztráción, titkárságon dolgozik nő. A felvettek körében a lányok aránya 10–15% körül van. Nemzetközi adatokat nem néztem, de a nemzetközi kampányok alapján egész biztos, hogy a lányok aránya nagyon messze van az 50%-tól és nagy szükség van az arányok javítására. Kérdés, hogy hogyan. Kérdés, hogy miért nem választják a lányok az informatikát, miért nem szeretnek programozni. Valóban nem szeretnek a lányok programozni? Kérdés, hogy hogyan lehet a lányok számára vonzóvá tenni az informatikus életpályát.

A fenti kérdésekre nem fogok itt válaszolni, de szeretném felhívni a figyelmet arra, hogy egy szakmában, ahol nagyon kevés a nő, az érvényesülni tudó nők nem biztos, hogy mérvadóak. Sokszor inkább kivételes esetnek számítanak, ezért óvatosan kell kezelni a sikerüket. Amikor egy nő azt mondja, hogy voltak nehézségek, de sikerült vennem az akadályokat, az nem azt jelenti, hogy megy ez, csak erősnek kell lenni, hanem azt, hogy férfiaknak találták ki a pályát. A kérdések egyik vetülete, hogy nők hiányában nehéz az együttműködés, hiányzik az empátia. Magas IQ-jú egyének nehezen kommunikálnak egymással, hiányzik az EQ. Ha egy ilyen közösségbe beteszünk egy magas EQ-jú egyént (mondjuk nőt), az számára egyáltalán nem lesz vonzó, mert a saját IQ-ját nem tudja érvényesíteni, ellenben az érzelmi intelligenciáját, empátiáját sokszorosan kihasználják a társak. Ez az állapot csak úgy viselhető el, ha az egyén alkalmazkodik, az érzelmeit kizárja és ő is csak az IQ érvényesítésére figyel.

Oktatásszervezési szempontból a fentihez hasonló helyzet elkerülésére javasolnám, hogy azt a kevés lányt, nőt, aki informatikus pályát választ, olyan közegben alkalmazzák, ahol érvényesülhet nőiségük. Például a 15%-os hallgatói arányt egyes csoportokra szétosztva, csoportonként 1–4 lány nem képes érvényesülni, különcök lesznek, akik alkalmazkodnak a fiúkhoz. Ezzel szemben olyan csoportok kialakítása, ahol a lányok legalább 40%-ban vannak jelen, lehetőséget adna számukra a női és szakmai kiteljesedésre is. Ezt ki kellene egészíteni azzal, hogy ezt a csoportot nők is oktassák.

Két példa az egyedüli nő helyzetére:

A legártatlanabb saját tapasztalatom a ZH-k javítása. A hallgatók sokszor írnak butaságot, ami javításkor az oktatókat hangos megjegyzésekre készteti. Értem én, de a stílusba néha már szinte belepirultam. Évek alatt megtanultam a védekezést: szó szerint veszem, ennek megfelelően rákérdezek arra, hogy hogyan gondolta az illető. De ha egyedül vagyok – márpedig ez a jellemző – akkor a hatás legfeljebb 5 perc. Egy nő kedvéért nem fogja egy csoport moderálni magát.

A hallgatók körében tapasztalható, hogy ha egy lány van a csoportban, akkor az vagy egyedül van vagy a barátjával (fiújával). Nagyon rizikós együtt dolgozni egy lánynak több fiúval, mert a fiúk jó eséllyel elkezdenek versengeni.

Oktatásmódszertan szempontjait nézve, a lányok genetikailag és nevelésük alapján is mást preferálnak, mint a fiúk. A lányok babáznak, a fiúk harcolnak. A lányok beszélnek, szóban támadnak, a fiúk mozgékonyabbak, ütnek. A lányok ruhát, szobát, bevásárlást, míg a fiúk fegyvert, várat, mentési akciót terveznek. A lányoknak, illetve a fiúknak ezért más tartalmú feladatok lesznek igazán érdekesek, motiválók. Tapasztalatom alapján, ez a fajta különbség a nevelésen, társadalmi elvárásokon alapul, ami a gyerek 2–3 éves korától (sőt, születése előtt 4 hónappal) kezdődik.

A probléma megoldásához hozzáállás oka lehet, hogy genetikai is, de közrejátszanak társadalmi, vallási elvárások is. A lányok inkább megtanulják, magolják először az ismereteket és ezután próbálják ki – vagy nézik, hogy mások hogyan csinálják meg, ezzel szemben a fiúk hajlamosabbak arra, hogy először kipróbáljanak valamit, és ha gond adódik, akkor kezdjenek utánajárni az irodalomban. A lányokra a megvizsgál, mérlegel, megjegyez, tervez tevékenységek (ebben a sorrendben) jellemzőbbek, míg a fiúkra a szétszed, megvizsgál, tapasztalatgyűjtés tevékenységek jellemzőbbek. A lányok a biztonságot hátra húzódva keresik, a fiúk kiharcolják maguknak, megvédik a területüket. A lányokra jellemzőbb, hogy ha csak félig-meddig tudnak valamit, akkor nem írják le, nem csinálják meg, míg a fiúkat könnyebb rávenni arra, hogy csináljon hasonlót.

A lányok sokkal inkább érzelmi alapon kezelik a tanulást. Sokkal többet megtanulnak, bemagolnak azért, hogy „jók” legyenek, mint a fiúk. A fiúkat kevésbé érdekli a környezetük, felettesük véleménye, azzal foglalkoznak és azt tanulják meg, ami érdekli őket.

Példa fiú és lány feladatmegoldására:

Egykori leány tanítványom külföldi részképzésen vett részt, ahol egy fiúval közösen kellett megoldaniuk a feladatot. A fiú azonnal elkezdte a megoldást, ő dolgozott, a lány

csak olvasgatott. Elolvasta az ajánlott forrásokat. Egy idő után a fiú valamit rosszul csinált, de nem tudta, hogy hol rontotta el. A lány megnézte, hogy mi a hiba és emlékezett arra, hogy erről olvasott valamit. Meg is találta, kijavította a fiú munkáját.

A példát – és a tipikus gyakorlatot – másként nézve: egy informatikai feladat megoldása során, ha egy fiú és egy lány együtt dolgozik, akkor a fiú kommunikál a géppel, a lány kontrollálja, tanácsot ad… nem a géppel kommunikál, hanem a fiúval. A fiú önbecsülését az erősíti, hogy ő oldotta meg a feladatot, ezért a lány szerepét, tudását hangsúlytalanítja. A lány fizikailag „nem csinál semmit", csak beszél, beleszól. Tudását a fiún keresztül érvényesíti, de eközben nem csak az informatikai problémára kell figyelnie, hanem arra is, hogy a fiú elfogadja őt is, az érveit is, ha szükséges a bírálatot is.

A tanulás során, csoportmunkában meg lehet vitatni a lehetőségeket, lehet közös döntéseket hozni, de a végrehajtás, a kódolás, a dokumentum megformázása, a router beállítása egy személy feladata. A csoport egyik tagja fogja megvalósítani a közös döntést. A siker a kivitelezőé, neki tulajdonítják a csoport minden tudását is. Ellenben a kudarcban osztozik a csapattal, hiszen ott már felmerül a hiba forrásának a keresése.

Ha a csapatban két fiú vagy két lány van, akkor is a párosból nagyon valószínű, hogy mindig az egyik lesz a kivitelező, a másik a háttérmunkás. Idővel a kivitelező azt fogja mondani magáról, hogy tud programozni, tudja az informatikát – mert bizonyította már számtalan esetben –, a háttérmunkás pedig úgy fogja érezni, hogy nem tud programozni, mert mindig segítséggel készült el a megoldás.

A tudással kapcsolatos önértékelés és a feladatmegoldási stratégia erősíti egymást. A fiús képességtudat a feladat megoldásának átgondolás nélküli elkezdésével erősítik egymást. A lányok önbizalomhiánya azzal, hogy a feladatot akkor kezdik végrehajtani, ha látják a célig vezető utat, szintén erősítik egymást.

8.2.1

XIV/6. Diákok teljesítményének elemzése

A 6. táblázat és a 40. ábra mutatja azoknak az osztályoknak az eredményét, amelyekben lehetőségem volt programozást tanítani. Minden csoportot én tanítottam, a felhasznált értékelést én végeztem. Kivétel közöttük a 2012b_term6, amelynek többször volt tanárváltása. A kutatás során kontroll csoportként tekintettem rájuk.

6. táblázat: Néhány osztály témazáróinak eredménye

	elm	**rajz**	**szov**	**prez**	**tk alap**	**tk köz**	**tk ab**	**web**	**komm**	**doc+ stílus**	**ab sql**	**prog**
2011d_ált4	4,49	4,72	4,22	4,54	3,47	4,13	3,87					3,71
2011a_mat6	4,44	4,62	4,65	4,15	4,44	4,10	3,89	4,71		4,14	3,73	3,52
2012b_term6	4,29	4,61	4,38									
2015a_mat6	4,62		4,49	4,42		4,05		4,72	4,22	3,98		4,72
2016b_term6	4,34	4,61	4,59	4,42	4,09	4,39	4,24			3,72		4,27
2017b_term6	4,22	3,97	4,08	4,54	4,54	4,47		4,52		4,40		4,36
2017c_ált4	4,39	4,31	4,06	4,36	3,92	3,46		4,10	4,21	3,63	4,22	4,41

A „mindent megtanulós” kitűnő osztály

A 2012b_term6 hatévfolyamos, természettudományos képzésű osztály, akit 7. és 9. évfolyamon tanítottam. 8. osztályban kollégám elölről kezdte tanítani a szövegszerkesztést, majd félévet táblázatkezeléssel is foglalkoztak. 9. évfolyamon itt fordult elő, hogy egy diák azért panaszkodott, mert a dolgozatban nehezebb feladatot adtam – INDEX(HOL.VAN()) nehezebb, mint az INDEX(HOL.VAN(MAX())) –, ezért adta be üresen a dolgozatát. Ugyanakkor legalább „jó”-t várt tőle a szülő, mert egy szorgalmasan tanuló, sok ismerettel bíró diákról van szó.

Év végén újabb konfliktusom volt: a prezentációkészítés projektfeladata egy matematika tétel bizonyítása volt; a matematikából is kitűnő diák a Thalész-tétel bizonyítása során rosszul rajzolta be a kört, ezért levontam egy jegyet – „de miért rossz, ha neki így tanították…”. 10. osztályban ismét a kollégám tanította őket, kevés sikerrel újratanulták a táblázatkezelést. 11-12. évfolyamon új tanárt kaptak, ahol még kétszer tanulták a középszintű érettségi táblázatkezelés részét, programozásra sem idő, sem motiváció nem volt.

A kollégák arról panaszkodtak, hogy mindig újra kérik a magyarázatot a diákok, minden feladatot külön akarnak megtanulni, nem értenek semmit. Ettől függetlenül (mert a dolgozatok is igazodtak a színvonalhoz) nagyon sok kitűnő tanuló volt az osztályban. Az egyetemekre is felvették a diákokat, de a visszajelzések mutatják, hogy az egyetemen a befogadó tanulással nagyon nehéz előre jutni. Ez a csoport – két szakkörös diák kivételével – nem tanult programozni.

Ahol lehetett programozást tanítani

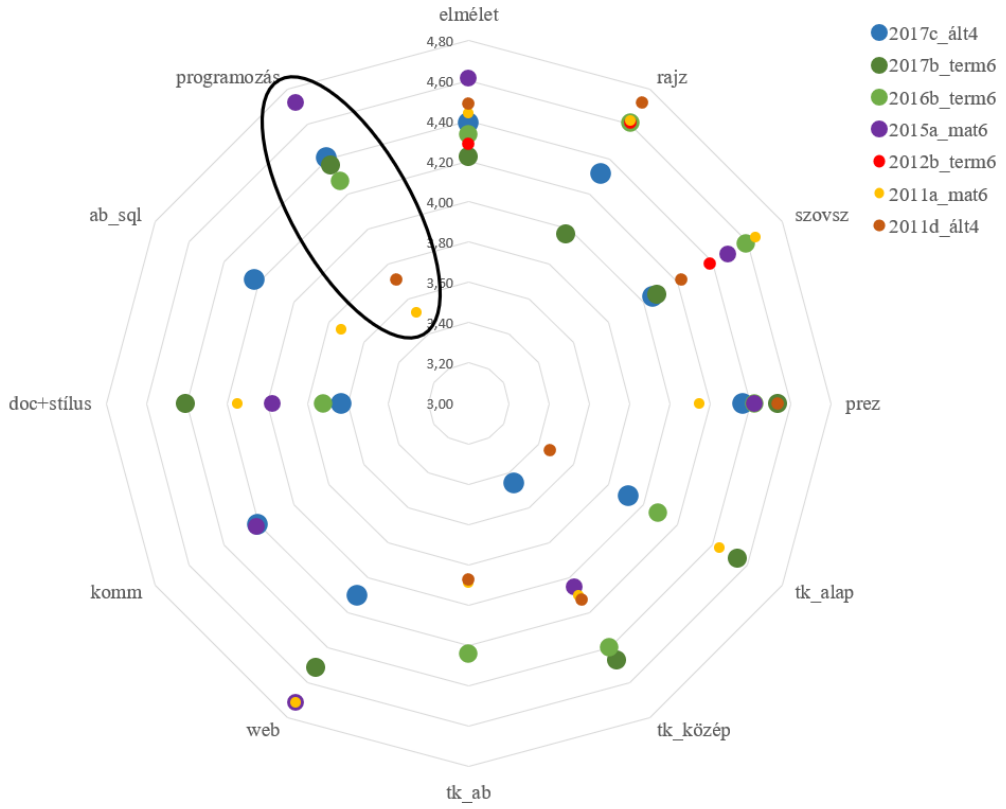
A 2011a_mat6 ugyanabban az időszakban hatévfolyamos matematika tagozatos osztály a programozást a 7–9 és 11. évfolyamon tanultak, a táblázatban a 9. osztályban írt programozás dolgozat eredménye látszik. (A 10. évfolyam – és a 2012b 9. évfolyam – jegyeit nem találtam.) Az informatika óraszámok náluk: 1+1+1+1+2+1. Ők 12. évfolyamon sokkal jobban tudtak már programozni, de a vizsgálat szempontjából a programozás „tipegő” szintje az, amit minden osztályban egységesen tudok vizsgálni, mivel az óraszámok idővel drasztikusan csökkentek.

A 2011d_ált4 osztály négyévfolyamos, általános tantervű osztály, a 9–10. osztályban, heti 2+1 órában tanultak láthatók a táblázatban. Náluk a programozás kitekintés volt, a továbbiakban nem szerepelt ez a téma.

A 2015a_mat6, 2016b_term6 és 2017b_term6 egymást követő hatévfolyamos képzésű matematika-, illetve természettudományi tagozatos osztályok. A matematikatagozaton 1+1+1+1+0+0 informatikaóra van, a természettudományi tagozaton 1+1+1+1+1+1, de mindhárom osztálynál csak a 7–9 évfolyamot néztem, a 9. év végén van a programozás „tipegő” szint (azaz 2018, 2019, 2020 tavaszán). Emellett a 2017c_ált4 osztály egyik csoportját tanítottam, 2+2+1 órában. Náluk 2020 tavaszán, 11. évfolyamon volt programozás.

Az eredmények háttere

Az elmélet (a tanulmányok legelső) dolgozat nagyon közeli eredményt mutat. Ez a dolgozat kérdéseiben és értékelésében is csak annyit változott, amennyit a technika fejlődött 10 év alatt. Az ezt követő dolgozatok már sokkal nagyobb szórást mutatnak. Egyre tudatosabban tanítottam a problémamegoldást a digitális írástudás témákban (rajz, szovsz, tk_alap), a magoláshoz szokott tanulók rossz jegyeket szereztek. Ha az igény a jó jegyre elég erős, akkor már a rajz dolgozat során rájönnek a diákok, hogy hogyan kell gondolkodni.



40. ábra: Osztályok, csoportok dolgozatainak eredménye

A táblázatkezelés jegyében az előtte levő témáknál erőteljesebb a problémamegoldás képességének súlya, a dolgozat eredménye jelzés arra, hogy a csoport milyen arányban gondolkodik, tud-e önállóan megoldást találni problémákra. Sokan ekkor élik meg a problémamegoldás képességét, de a csoport összetételétől, a kamaszodás mértékétől erősen függ az eredmény. Ilyenkor már világos a diákok jelentős része számára, hogy nem én fogok engedni a követelményekből, nincs „gyűjtőmunkával" helyettesítésre lehetőség…

Látható, hogy a 2015 utáni osztályok közül, amelyik csoport jó dolgozatot ír táblázatkezelésből, annak a programozás dolgozata is jó lesz. Pontosabban: lényegében név szerint tudom, hogy kik azok, akik megoldják a problémákat és kik azok, akik megtanulnak megoldásokat. A csoporton belüli arányuk tükröződik az átlagban.

A programozás dolgozat eredményén jól látszik a módszertani váltás:

- az algoritmusok, matematikai alapozás helyett a problémamegoldásra helyezett fókusz;
- az alkalmazások tanításába egyre tudatosabban beépített informatikai gondolkodás és ennek egyre tudatosabb számonkérése – akár a jegyek romlásának kockáztatásával is, ezzel összefüggésben a diákoktól a gondolkodás határozott, kérlelhetetlen elvárása;
- a programozás oktatása során a nem gondolkodók elkülönítése és gondolkodásra „kényszerítése".

Távoktatás hatása

A programozás tanulás elemzésére a távoktatás a megszokottnál jobb feltételeket adott. A diákok nehezebben kommunikáltak egymással vagy külső tanácsadókkal. Amikor mégis, az azonnal észrevehető volt a beadott munkáikon.

Három csoportban (2017b osztály és 2017c csoport), 51 diáknak a karantén előtti, kisebb megszakításokkal 8 programozás óra után körülbelül 10 tanórányi távoktatás következett, ahol a feladatokat és a jegyzeteket dokumentumok formájában adtam ki, a feladatok a http://mester.inf.elte.hu oldalról származtak. A témazáró feladat is ugyaninnen, korábban még nem nézett feladatokból volt.

A tanórákon online konzultációt tartottam, amelyen 1–5 diák vett részt, emellett bármikor lehetett üzenőfalon kérdezni. (Volt, hogy vasárnap éjfélkor is válaszoltam.) Egyes diákok ismerőstől, testvértől kértek segítséget – C# helyett Pythonban vagy láthatóan tanult ember megoldását adták be. (Például beolvasás ellenőrzése is volt benne.) Mások osztályon belül kértek tanácsot, így – más változónevekkel, de – egyforma megoldások születtek. Ezt mindig jeleztem, megbeszéltem az adott diákokkal, hogy mennyire veszélyes más gondolatát megtanulni.

A témazáróját 29 diáknak (56%) jelesre értékeltem. Ez azt jelenti, hogy ők a kiadott feladatot 45 perc alatt megkeresték, megoldották, a kódot tesztelték és az 100%-os lett, valamint a megoldást (kódot, képernyőképet) beadták, a kód tiszta rendezett volt.

A 7. táblázatban tanulmányozható a tapasztalt hibák és ezek értékelése. Látható, hogy további 7 diák (15%) apró hibával oldotta meg a feladatot. A többiek jellemzően a feladat megoldását elkezdték, volt működő része a programnak, de valahol belezavarodtak és nem volt elég idő a probléma megoldásra.

Négy diák jött az online dolgozat után be az iskolába személyes konzultációra, majd javító dolgozatra. Közülük hárman távoktatás alatt nem jutottak előbbre. A dolgozatot a konzultáció után 2 nappal, szintén az iskolában felügyelet mellett írták, néha segítséggel, amit a dolgozat értékéből levontam.

Egy diák feladatát más oldotta meg (és tesztelte is a portálon), Ő elégtelent kapott. A felkészülés ideje alatt nyújtott teljesítménye alapján önállóan 2-3-as dolgozatot tudott volna írni, lustaságból következő rutinhiány miatt. Egy diák – aki fakultációra is járt, ezért egy érettségi feladatot kellett volna félig megoldania – a tavaszi szünetet követően „eltűnt”. A távoktatás felerősítette az önbizalomhiányát.

7. táblázat: A 2020-as karantén idején programozást tanulók nem hibátlan dolgozatainak értékelése

kód	jegy	feladat	hiba
9bs08	4,8	Ki30	formátumhibás a kiírás
9bs10	4,8	Ki28	spec. esetre nem figyel
9bs14	4,8	Ma25	maxert = 0 - spec. eset
11c06	4,5	Ke15	időtúllépéssel javított elírás
9bs12	4,5	Ma25	osztásban int a double helyett
9bs17	4,5	Ke10	hibás tagadás (és\vagy)
9bs18	4,5	Ma03	kishibás bázisváltás
9bl06	4	Ki06	hibás kiírás
9bs05	4	Me06	elbonyolított, nincs kiírás
11c04	3,5	Me45 / El05	segítséggel: először be kell olvasni
11c19	3,5	Ke11 / Me39	segítséggel: az elírások javítása
9bl11	3,5	Me45	tömb túlindexelése
9bs15	3,5	Ma11	előző elemhez viszonyít maximum keresésekor
9bs19	3	Ke15 / Ma17	iskolában, sok bíztatással: eddig jó.
9bl02	2,5	Me38	hibás feltöltés, hibás algoritmus
11c08	2	Ke20	bonyolult hibás algoritmus, pontatlan beolvasás
11c09	2	Me06	jó beolvasás "==<" jelölés
9bl07	2	Ma32	i>b, fordított értékadás
9bl08	2	Ke18	beolvasás jó
9bl17	1	Ke11	más oldotta meg.
11c07	-	Érettségi	eltűnt diák

Utómérés

A 2016b osztálynak 2021-ben is volt informatikaórája. Tanév végén kilenchetes projekt feladatot kaptak: egyénileg választott témához esszédolgozat (stílus alkalmazása és tartalomjegyzék is), prezentáció (értelmezést támogató animációval), weblap (több lap, html és css online források alapján), táblázat (feltételfüggő számítással, kereséssel), adatbázis (adatok gyűjtése, többtáblás adatbázis megtervezése, lekérdezések) és program készítése (bekérés, kiírás feltételek vagy ciklus; például teszt; a „tipegő” szint „jó”). A témákra részjegyeket kaptak, de végül a hat témából csak az öt legjobb számított be a végső osztályzatba.

Az egyes témakörökre számított statisztikai adatok a 8. táblázatban láthatók.

8. táblázat: A 2021 tanév végi projekt (szintfelmérő) eredménye

	Szöveg-szerkesztés	Prezen-táció	Weblap, HTML	Táblázat-kezelés	Adatbázis-kezelés	Program-készítés
Szint (érettségi)	emelt	emelt	közép	emelt	közép + DDL	„tipegő”
Átlag	4,06	4,33	4,23	4,03	3,71	3,97
Szórás	1,07	0,98	1,01	1,32	1,48	1,31
Elégtelen (db)	1	1	1	2	3	1
Jeles (db)	13	17	15	16	16	17

Az eredmények értelmezéséhez hozzá tartozik, hogy a projektidőszak első fele online, második fele kontakt óra volt. Az átállás és a tanév vége több diáknak több tantárgyból extra terhelést jelentett. Az osztály jelentős része biológia-kémia fakultációt választott, informatika fakultációra nem jelentkeztek. Az informatika – heti 1 tanóra két csoportban – pénteken volt, távoktatásban a 3. és 4. órában, iskolai jelenléttel az 1. és 7. órában.

8.4.

Felhasznált irodalom és források

1. KOPLÁNYI Emil: Magyar Soros Alapítvány Informatikai fejlesztései a közoktatásban. Sorozatcím: Programértékelő kiadvány (diplomamunka) Soros Alapítvány Budapest (2000);
   Megtekintés 2021.08.21. http://kka.hu/_soros/kiad-vany.nsf/daaf25ff08ec8dd1c1256e9f00417f81/881742c5b2b48d93c1256e47003b07a3
2. KOMENCZI Bertalan: Informatizált tanulási környezetek fejlesztése – Doktori értekezés (2003); Megtekintés: 2021.08.21.
   https://repozitorium.omikk.bme.hu/bitstream/handle/10890/183/ertekezes.pdf
3. TOMPA Klára (szerk): Az érettségiről tanároknak – 2005 – Informatika. OFI Budapest, (2004); Eredeti hely (megszűnt): http://www.okm.gov.hu/letolt/kozokt/erett-segi2005/tanaroknak/main/fomenu.htm. Publikus archívumban megtekintés: 2021.08.21.
   https://docplayer.hu/4337179-Reszlet-az-erettsegirol-tanaroknak-informatika.html
4. E. SOLOWAY: Should we teach students to program? In: Communications of the ACM, Vol 36 No 10, p: 21–24 (1993). DOI: 10.1145/163430.164061. Megtekintve: 2021.08.21. https://dl.acm.org/doi/10.1145/163430.164061
5. Marc PRENSKY: Digital Natives, Digital Immigrants. In: MCB University Press, Vol 9 No 5, (2001) Megtekintés: 2021.08.21. https://marcprensky.com/writing/Prensky%20-%20Digital%20Natives,%20Digital%20Immigrants%20-%20Part1.pdf
6. Paul A. KIRSCHNER, Pedro DE BRUYCKERE: The myths of the digital native and the multitasker. In: Teaching and Teacher Education. Vol 67, p: 135–142 (2017) DOI: 10.1016/j.tate.2017.06.001 Megtekintés: 2021.08.22. https://www.rese-archgate.net/requests/attachment és https://www.gwern.net/docs/psychology/2017-kirschner.pdf
7. Barbara ERICSON, Michal ARMONI, Judith GAL-EZER, Deborah SEEHORN, Chris STE-PHENSON, Fran TREES: Ensuring Exemplary Teaching in an Essential Discipline: Ad-dressing the Crisis in Computer Science Teacher Certification – Final Report of the CSTA Teacher Certification Task Force. ACM Press, (2008), ISBN: 9781605583914; Megtekintés 2021.08.21. https://www.researchgate.net/publication/304674511
8. Päivi KINNUNEN: Challenges of Teaching and Studying Programming at a University of Technology – Viewpoints of Students, Teachers and the University. Helsinki University of Technology Faculty of Information and Natural Sciences Department of Computer

Science and Engineering, Finland (2009). ISBN: 9789522481955. Megtekintés: 2021.08.21. http://lib.tkk.fi/Diss/2009/isbn9789522481955/isbn9789522481955.pdf

9. Csíkszentmihályi Mihály: Flow. Az áramlat - A tökéletes élmény pszichológiája. (1991) Ford. Legéndyné Szabó Edit. Akadémiai Kiadó Bp. (2001) ISBN: 9630577704
10. SZLÁVI Péter, ZSAKÓ László: Mi az informatika szakmódszertan? In: Informatika oktatása. ELTE IK, (2012); Megtekintés: 2021.08.21: https://regi.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0052_34_informatika_oktatasa/2011-0052_34_informatika_oktatasa.zip
    Megtekintés: 2021.10.29: https://dtk.tankonyvtar.hu/handle/123456789/3843
11. FALUS Iván (szerk): Didaktika – Elméleti alapok a tanítás tanulásához,
    NTK, Budapest (1998); ISBN: 9631890759
    Digitális Tankönyvtár (forrás: NTK 6. kiadás) (2003); ISBN: 9631944557;
    https://regi.tankonyvtar.hu/hu/tartalom/tamop425/2011_0001_519_42498_2/2011_0001_519_42498_2.pdf
    Részlet: KOTSCHY Beáta: Az oktatás célrendszere. p: 118. (…/ch06s05.html)
    Megtekintés: 2021.08.21. (2021.09.01-től https://dtk.tankonyvtar.hu/ ?)
12. BREDÁCS Alice Mária: A hagyományos és az IKT-vel támogatott mérés és értékelés a szakképzésben Digitális Tankönyvtár (BME Tanárképző Központ) (2015);
    Megtekintés: 2021.08.21. https://regi.tankonyvtar.hu/hu/tartalom/tamop412b2/2013-0002_a_hagyomanyos_es_az_ikt-vel_tamogatott_meres_es_ertekeles_a_szakkepzesben/HI/shijs23g.scorm (2021.09.01-től https://dtk.tankonyvtar.hu/ ?)
13. John B BIGGS, Kevin F. COLLIS: Evaluating the Quality of Learning – The SOLO Taxonomy. Academic Press Inc. (1982); ISBN: 9780120975525;
    DOI: https://doi.org/10.1016/C2013-0-10375-3. p: 24–25. Betekintés: 2021.08.21.
    https://books.google.hu/books?id=xUO0BQAAQBAJ&lpg=PP1&ots=aqlAhYPuOc&dq=%22Evaluating%20the%20Quality%20of%20Learning%3A%20The%20SOLO%20Taxonomy%22&lr&hl=hu&pg=PA23#v=onepage&q&f=false
14. Dorian LOVE: Why I prefer the SOLO Taxonomy to Bloom's; The DigiTeacher (Using Technology in Education to foster critical thinking) blog bejegyzés (2014.04.11). Megtekintés: 2021.08.21. https://digiteacher.wordpress.com/2014/04/11/why-i-prefer-the-solo-taxonomy-to-blooms/

15. Russel L. ACKOFF: From Data to Wisdom. In: Ackoff's Best (His Classic Writings on Management); John Wiley & Sons New York; (1999); ISBN: 0471316342; p: 170–172; Megtekintés: 2021.08.21. http://softwarezen.me/wp-content/uploads/2018/01/datawisdom.pdf
16. Jay H. BERNSTEIN: The Data-Information-Knowledge-Wisdom Hierarchy and its Antithesis In: Proceedings from North American Symposium on Knowledge Organization ISSN: 2311-4487; (2009) Vol 2, p: 68–75; DOI: 10.7152/nasko.v2i1.12806. Megtekintés: 2021.08.21. https://journals.lib.washington.edu/index.php/nasko/article/viewFile/12806/11288
17. Stan GARFIELD: Yet Another Myth: The DIKW Pyramid Scheme; https://www.linkedin.com/pulse/yet-another-myth-dikw-pyramid-scheme-stan-garfield (2015.11.06) https://medium.com/@stangarfield/yet-another-myth-the-dikw-pyramid-scheme-a059ba595b30 (2018.03.25) Megtekintés: 2021.08.21.
18. MAKÓ Ferenc: 1.2 A tanulás, mint információ felvevő-feldolgozó és előhívó folyamat. In: Tanulásmódszertan, Óbudai Egyetem (2015.09.19). Megtekintés: 2021.08.21. https://www.tankonyvtar.hu/hu/tartalom/tamop412b2/2013-0002_tanulasmodszertan/tananyag/JEGYZET-05-1.2._A_tanulas__mint_informac.scorml Megtekintés: 2021.10.29.: https://dtk.tankonyvtar.hu/handle/123456789/3892 (kitömörítés után megtalálható a kép)
19. Edgar DALE: The cone of experience. In: Audio-visual methods in teaching. New York, Dryden Press. (1969) p: 37–51.
20. STEPS for Pharmacy Educators: Dale's Cone of Experience. Megtekintés: 2021.08.21. https://www.queensu.ca/teachingandlearning/modules/active/documents/Dales_Cone_of_Experience_summary.pdf
21. Scott SMITH: How to Maximize the Retention and Comprehension of Course Content Material in Training. Megtekintés: 2021.08.21. https://www.slideshare.net/ScottSmith/Active-learning-1327412 (2009.04.22)
22. Sue SENTANCE: PRIMM: A structured approach to teaching programming. In blog: Computer Science Education @ King's; Megtekintés: 2021.08.21. https://blogs.kcl.ac.uk/cser/2017/09/01/primm-a-structured-approach-to-teaching-programming/ (2017.09.01)

23. Sue SENTANCE: Exploring pedagogies for teaching programming in school. In blog: Computer Science Education @ King's; Megtekintés: 2021.08.21. https://blogs.kcl.ac.uk/cser/2017/02/20/exploring-pedagogies-for-teaching-programming-in-school/ (2017.02.20)
24. Jaap M.J. MURRE, Joeri DROS: Replication and Analysis of Ebbinghaus' Forgetting Curve. In: PLoS One Vol 10 No 7; (2015); DOI: 10.1371/journal.pone.0120644; Megtekintés: 2021.08.21. permalink: https://hdl.handle.net/11245/1.503084 final: https://pure.uva.nl/ws/files/2701916/172027_503084.pdf
25. John SWELLER: Cognitive Load Theory. In: Jose P. Mestre, Brian H. Ross (szerk.): Psychology of Learning and Motivation; Academic Press (2011), Vol 55. p: 37–76. ISSN 0079-7421; ISBN 9780123876911; DOI: 10.1016/B978-0-12-387691-1.00002-8.
26. D. KAHNEMAN: Thinking, Fast and Slow, New York: Farrar, Straus; Giroux. (2011). ISBN: 9780374533557, DOI: 10.19232/uv4pb.2016.1.92
27. D. KAHNEMAN, A. TVERSKY: Prospect Theory: An Analysis of Decision Under Risk (Abstract). In: Handbook of the Fundamentals of Financial Decision Making part I. ch. 6. p:99–127 World Scientific, (2013). DOI: 10.1142/9789814417358_0006. Megtekintés: 2021.08.26. http://hassler-j.iies.su.se/COURSES/NewPrefs/Papers/KahnemanTversky%20Ec%2079.pdf
28. J A CHEN: Implicit theories, epistemic beliefs, and science motivation: A person-centered approach. In Learning and Individual Differences, Vol 22 No 6, 724–735. DOI: 10.1016/j.lindif.2012.07.013 Megtekintés: 2021.08.27. https://core.ac.uk/download/pdf/235396776.pdf
29. J. A. CHEN, D. B. MORRIS, N. MANSOUR: Science Teachers' Beliefs. Perceptions of Efficacy and the Nature of Scientific Knowledge and Knowing. In: Fives, Gill, Routledge: International Handbook of Research on Teachers' Beliefs. p: 370–386. (2015) Megtekintés: 2021.08.22. https://scholarworks.wm.edu/bookchapters/2 https://scholarworks.wm.edu/cgi/viewcontent.cgi?article=1002&context=bookchapters
30. J. HATTIE: Visible learning for teachers: Maximizing impact on learning. Routledge/Taylor & Francis Group (2012).
31. Richard R. SKEMP: Relational Understanding and Instrumental Understanding. In: Mathematics Teaching, Vol 77 p. 20–26 (1976) Megtekintés: 2021.08.29. https://www.atm.org.uk/write/mediauploads/resources/richard_skemp.pdf

32. POLYA, G. (1957) How to Solve It. A New Aspect of Mathematical Method. 2nd Edition, Princeton University Press, Princeton. Megtekintés: 2021.08.21. https://lms.umb.sk/pluginfile.php/37176/mod_folder/content/0/Polya_How-to-solve-it.pdf szkennelt: https://pdfcoffee.com/qdownload/george-polya-how-to-solve-it-pdf-free.html
33. LÉNÁRD Ferenc: A problémamegoldó gondolkodás. Akadémiai Kiadó, Budapest (1984)
34. S. PAPPERT: The Children's Machine: Rethinking School in the Age of the Computer. BasicBooks, New York (1993). ISBN: 0465018300
35. KOVÁCS András: A matematikai problémamegoldás iskolai alkalmazásairól. (PhD értekezés) Debreceni Egyetem, (2006) Megtekintés: 2021.08.21. https://dea.lib.unideb.hu/dea/bitstream/handle/2437/78506/de_3354.pdf
36. KONTRA József: A probléma és a problémamegoldó gondolkodás. In: Magyar Pedagógia Vol 96 No 4, p: 341–366. (1996) http://www.magyarpedagogia.hu/document/Kontra_MP964.pdf
37. Jeannette M. Wing: Computational Thinking. Communication of the ACM Vol 49 No 3, p: 33–35 (2006) DOI: 10.1145/1118178.1118215 Megtekintés: 2021.08.22. https://www.cs.cmu.edu/~15110-s13/Wing06-ct.pdf
38. Jeanette M. WING: Computational Thinking: What and Why? (2010.11.17.) Megtekintés: 2021.08.23. https://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf
39. Pluhár Zsuzsa: Az informatikai gondolkodás és a hód. In: Szlávi Péter, Zsakó László (szerk): INFODIDACT'2016, Webdidaktika Alapítvány (2017) ISBN: 9786158060806 Megtekintés: 2021.08.22. https://people.inf.elte.hu/szlavi/InfoDidact16/Manuscripts/PZs_Hod.pdf
40. Thomas GORDON: T.E.T – A tanári hatékonyság fejlesztése. Assertiv Kiadó, (1998)
41. U. NIKULA, O. GOTEL és J. KASURINEN: A Motivation Guided Holistic Rehabilitation of the First Programming Course. ACM Transactions on Computing Education, Vol 11 No 4, (2011). Megtekintés: 2021.08.29. https://dl.acm.org/doi/pdf/10.1145/2048931.2048935?casa_token=g-kxIe-UY90AAAAA:f-Cg3Erbe3jjlRQ2R1YwJEl0YnS-TiYQe7v-vidxngkDEINEybfPQ8Pg0X7GPBpR2h-FAFlPOcX1EQ
42. MAKÓ Ferenc: Tanulásmódszertan. Óbudai Egyetem (2015); Megtekintés: 2021.10.26. https://dtk.tankonyvtar.hu/xmlui/handle/123456789/3892

43. European Political Strategy Centre (European Commission): 10 trends transforming education as we know it. European Union (2017). ISBN: 9789279753473, DOI: 10.2872/800510. Megtekintés: 2021.08.21. https://op.europa.eu/en/publication-detail/-/publication/227c6186-10d0-11ea-8c1f-01aa75ed71a1 (2019.11.20)
44. Kalelioğlu, FILIZ, Yasemin GÜLBAHAR, Dilek DOĞAN: Teaching How to Think Like a Programmer: Emerging Insights. In: Huseyin OZCINAR, Gary WONG, H. Tugba OZTURK (eds): Teaching Computational Thinking in Primary Education. Hershey, IGI Global p: 18–35, (2018). ISBN: 9781522532002, DOI: 10.4018/978-1-5225-3200-2.ch002 Megtekintés: 2021.08.22. https://www.researchgate.net/publication/320878088_Teaching_How_to_Think_Like_a_Programmer_Emerging_Insights
45. George E. FORSYTHE: What to Do Till the Computer Scientist Comes. In: The American Mathamatican Monthly Vol 75 No 5, (1968); p. 454–462; DOI: 10.2307/2314698. Megtekintés 2021.08.22. előfizetéses: https://www.jstor.org/stable/2314698 vagy reprint https://stacks.stanford.edu/file/druid:rz632tc5340/rz632tc5340.pdf
46. Donald KNUTH: Computer Science and Its Relation to Mathematics. In: The American Mathematical Monthly Vol 81 No 4, (1974); p. 323–343 DOI: 10.2307/2318994; Megtekintés 2021.08.22. előfizetéses: https://www.jstor.org/stable/2318994 vagy free: https://www.maa.org/sites/default/files/pdf/upload_library/22/Ford/DonaldKnuth.pdf
47. Shaoyi HE: Informatics: a brief survey. In: The Electronic Library, Vol 21 No 2, (2003); p. 117–122, Fizetős elérhetőség: https://doi.org/10.1108/02640470310470480 Magyar interpretáció: [48]
48. PAPP László (ford), HE Shaoyi: Az informatika fogalma. In: Tudományos és Műszaki Tájékoztatás, Vol 50 No 9–10 (2003); Megtekintés: 2021.08.22. https://tmt.omikk.bme.hu/tmt/article/view/2070/3096
49. Informatika. Wikipédia szócikk Megtekintés: 2021.08.23. https://hu.wikipedia.org/wiki/Informatika (2021. augusztus 2., 09:27)
50. Edsgar W. DIJKSTRA: Why numbering should start at zero. (EWD831) (1982.08.11.) in: E. W. Dijkstra Archive Megtekintve: 2021.08.23. http://www.cs.utexas.edu/users/EWD/ewd08xx/EWD831.PDF vagy html formátumban https://www.cs.utexas.edu/users/EWD/transcriptions/EWD08xx/EWD831.html

51. NÉMETH Ádám: Számkörbővítés és műveletek az iskolában (szakdolgozat, matematikatanári szak). Budapest, ELTE TTK 2017. Megtekintve: 2021.08.23. https://web.cs.elte.hu/blobs/diplomamunkak/mattan/2017/nemeth_adam.pdf

52. BAKÁNYI Márton, FODOR Erika, MARX György, SARKADI Ildikó, TÓTH Eszter, UJJ János: Fizika gimnázium I. osztály p. 6. Tankönyvkiadó, Budapest, (1983)

53. SZALAYNÉ TAHY Zsuzsanna, CZIRKOS Zoltán: Az Informatika Más… In: KERESZTES Gábor (szerk.) Tavaszi Szél 2015 / Spring Wind 2015 Konferenciakötet: Vol III, Eger, Magyarország, Líceum Kiadó, (2015) p. 207–220. ISBN: 9786155509964, DOI: 10.17048/TSZ.2015.3; Megtekintés: 2021.08.23. https://www.dosz.hu/__doc/dokumentumfile/tsz2015_3.pdf

54. SZALAYNÉ TAHY Zsuzsanna, CZIRKOS Zoltán: "ProgAlap" és ami mögötte van. In: INFODIDACT 2015. Webdidaktika Alapítvány, (2015); p. 1–16 (#14) ISBN: 9789631238921 Megtekintés: 2021.08.23. https://people.inf.elte.hu/szlavi/InfoDidact15/Manuscripts/SzTZsCzZ.pdf

55. Edsgar W. DIJKSTRA: A Case against the GO TO statement (letter EDW 215) In Commun, ACM (11) (1968); 3, p. 147–148. Megtekintés: 2021.08.23. https://www.cs.utexas.edu/users/EWD/ewd02xx/EWD215.PDF Kapcsolódó: EWD1380 What led to "Notes on Structured Programming" https://www.cs.utexas.edu/users/EWD/ewd13xx/EWD1308.PDF

56. Robert C. MARTIN: Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall PTR, USA (2008) ISBN: 9780132350884, (megrendelés: https://dl.acm.org/citation.cfm?id=1388398)

57. SZALAYNÉ TAHY Zsuzsanna, CZIRKOS Zoltán: Linear search – the breaks in teaching-learning practice. In: New methods and technologies in education and practice – XXIX. DIDMATTECH 2016 Budapest, Magyarország: ELTE Informatikai Kar, (2016); p. 66–70. ISBN 9789632848167, Megtekintés: 2021.08.23. http://didmattech.inf.elte.hu/2016/wp-content/uploads/2016/08/SzZZs_CZ_Linear-Search.pdf

58. SZALAYNÉ TAHY Zsuzsanna, CZIRKOS Zoltán: A lineáris keresés buktatói. In: ZSAKÓ, László; SZLÁVI, Péter (szerk.) INFODIDACT 2016: Informatika Szakmódszertani Konferencia; Zamárdi, Magyarország: Webdidaktika Alapítvány, (2016); Paper: SzTZs_CZ, 10 p. ISBN 9786158060806. Megtekintés: 2021.08.23. https://people.inf.elte.hu/szlavi/InfoDidact16/Manuscripts/SzTZsCZ.pdf

59. SZALAYNÉ TAHY Zsuzsanna, CZIRKOS Zoltán: Így írtok Ti… lineáris keresést. In: ZSAKÓ, László (szerk.) INFO ÉRA 2016 Zamárdi, Magyarország: Webdidaktika Alapítvány, (2016); Megtekintés: 2021.08.23.
Paper: http://infoera.hu/upload/2016_1128_IgyIrtokTi.pptx

60. HORVÁTH Győző, MENYHÁRT László, ZSAKÓ László: Egy webes játék készítésének programozásdidaktikai szempontjai. In: INFODIDACT 2015. Webdidaktika Alapítvány, (2015); p. 1–10 (#2) ISBN: 9789631238921
https://people.inf.elte.hu/szlavi/InfoDidact15/Manuscripts/HGyMLZsL.pdf

61. E. W. Dijkstra Archive – the manuscripts of Edsgar W. Dijkstra (2017.10.31) Megtekintve: 2021.08.23. http://www.cs.utexas.edu/~EWD/

62. CZIRKOS Zoltán: Melyik rendezés? – Kódolvasási készségek fejlesztése. In: ZSAKÓ László (szerk.) INFO ÉRA 2017 Zamárdi, Magyarország: Webdidaktika Alapítvány; (2017)
Paper: https://infoera.hu/upload/2017_1210_czirkos-melyik_rendezes.pdf

63. KOVÁCS Györgyi, ROZGONYI-BORUS Ferenc: Az informatika oktatás története. In: In(formatika)Lap (2001.09.14.) Megtekintés: 2021.08.23. http://www.abax.hu/inlap/t/cikk/inftori.htm (2021.11.28.: Szerver megszűnt, költözés alatt)

64. 130/1995. (X.26) Korm. rendelet a Nemzeti alaptanterv kiadásáról (1995) Hozzáférés: 2021.10.24 http://www.jogiportal.hu/view/130-1995-x-26-korm-rendelet

65. 243/2003. (XII. 17.) Korm. rendelet a Nemzeti alaptanterv kiadásáról, bevezetéséről és alkalmazásáról Hozzáférés: 2021.10.24 http://www.nefmi.gov.hu/letolt/kozokt/nat_070926.pdf

66. 202/2007. (VII.31.) Korm. rendelet a Nemzeti alaptanterv kiadásáról, bevezetéséről és alkalmazásáról szóló 243/2003. (XII. 17.) Korm. rendelet módosításáról In: Magyar Közlöny 2007/102. Hozzáférés: 2021.10.24 http://www.kozlonyok.hu/nkonline/MKPDF/hiteles/MK07102.pdf

67. 110/2012. (VI.4) Korm. rendelet A Nemzeti alaptanterv kiadásáról, bevezetéséről és alkalmazásáról. In: Magyar Közlöny Magyarország, 2012.06.04. (2012) 66. szám p: 10635-10847 Hozzáférés: http://kozlony.magyarorszag.hu/dokumentumok/f8260c6149a4ab7ff14dea4fd427f10a7dc972f8/megtekintes

68. 5/2020 (I.31) Korm. rendelet a Nemzeti alaptanterv kiadásáról, bevezetéséről és alkalmazásáról szóló 110/2012 (VI.4) Korm. rendelet módosításáról. In: Magyar Közlöny

2020/17. Hozzáférés: 2021.10.24 https://magyarkozlony.hu/dokumentumok/3288b6548a740b9c8daf918a399a0bed1985db0f/letoltes

69. 51/2012. (XII. 21.) EMMI rendelet A kerettantervek kiadásának és jóváhagyásának rendjéről In: Magyar Közlöny 2012/177. Hozzáférés: 2021.10.24. http://www.kozlonyok.hu/nkonline/MKPDF/hiteles/MK12177-2R.pdf

70. Kerettantervek a 2020-as NAT-hoz
Digitális kultúra 3-4. évfolyam, Kerettanterv az általános iskola 1-4. évfolyam számára
Digitális kultúra 5-8. évfolyam, Kerettanterv az általános iskola 5-8. évfolyam számára
Digitális kultúra 9-11. évfolyam, Kerettanterv a gimnáziumok 9-12. évfolyam számára
In: A 2020-as NAT-hoz illeszkedő tartalmi szabályozók Hozzáférés: 2021.10.24
https://www.oktatas.hu/kozneveles/kerettantervek/2020_nat

71. SZŰCS Ervin, KERESZTESI Miklós: Technika I. Tankönyvkiadó Budapest (1981); ISBN: 963175622x és
Technika II. Tankönyvkiadó Budapest 1982 ISBN: 9631762106

72. BORSÁNYI Katalin, HACK Frigyes.: Matematika Feladatgyűjtemény III., 3 szerk., Vol III., Budapest: Tankönyvkiadó, (1990).

73. ZSAKÓ László: Miért elavult informatikából a NAT, a kerettanterv, az érettségi? INFODIDACT 2015, Informatika Szakmódszertani Konferencia; Zamárdi. [74] prezentációja (2015) Megtekintés: 2021.08.23. https://slideplayer.hu/slide/2902432/

74. ZSAKÓ László: Informatika Nemzeti Alaptanterv 2020. In: Dr. Szlávi Péter, Dr. Zsakó László (szerk): INFODIDACT 2015, Informatika Szakmódszertani Konferencia; Zamárdi, Magyarország: Webdidaktika Alapítvány, (2015); Paper: ZsL.pdf, 10 p. ISBN 9789631238921 Megtekintés: 2021.08.23.
https://people.inf.elte.hu/szlavi/InfoDidact15/Manuscripts/ZsL.pdf

75. DANCSÓ Tünde, KOROM Pál: Informatika 9-10. A gimnáziumok számára (NT-17173). Nemzedékek Tudása Tankönyvkiadó (Oktatáskutató és Fejlesztő Intézet), Budapest (2013).

76. BÁNHIDI Sándorné, FÜLÖP Márta Marianna, GYARMATHY Éva: Az informatika interdiszciplináris dimenzióinak tudományos megalapozását és megújítását segítő kutatás eredményei. Informatika-Számítástechnika Tanárok Egyesülete (az Oktatáskutató és Fejlesztő Intézet megbízásából) Budapest (2015); Megtekintés: 2021.08.23.
https://isze.hu/download/Zarodokumentum_ISZE_kutatas.pdf

77. SZALAYNÉ TAHY Zsuzsanna, CZIRKOS Zoltán: "Why Can't I Learn Programming?" The Learning and Teaching Environment of Programming. In: ISSEP 2016; LNCS Vol 9973; Springer International Publishing; p. 199–204., (2016); ISBN: 9783319467474; DOI: 10.1007_978-3-319-46747-4_17

78. TORMA Hajnalka: Computational thinking - could we bring it to Hungarian education? In: Veronika Stoffová, Zsakó László, Szlávi Péter (szerk.): New methods and technologies in education and practice: Proceedings of XXIX. DIDMATTECH 2016. Budapest: ELTE Informatikai Kar, (2016) p. 117–121. ISBN 9789632848167, Megtekintve: 2021.08.26. http://didmattech.inf.elte.hu/2016/wp-content/uploads/2016/08/TH_Computational-thinking.pdf

79. Finnish National Board of Education: New national core curriculum for basic education: focus on school culture and integrative approach. (2016) Megtekintve: 2021.10.24. https://www.oph.fi/sites/default/files/documents/new-national-core-curriculum-for-basic-education.pdf

80. Department for Education, GOV.UK: „National curriculum," 16. július 2014. Megtekintés: 2021.10.24. https://www.gov.uk/government/collections/national-curriculum

81. Françoise TORT, François-Marie BLONDEL, Éric BRUILLARD: Spreadsheet Knowledge and Skills of French Secondary School Students. In: ISSEP 2008, Torun, Poland; Proceedings. LNCS 5090, Springer 2008; p. 305–316 (2008) ISBN: 9783540699231 Megtekintés: 2021.08.29. https://www.researchgate.net/publication/221437637_Spreadsheet_Knowledge_and_Skills_of_French_Secondary_School_Students/citations

82. Natasa GRGURINA, Jos TOLBOOM: The First Decade of Informatics in Dutch High Schools. In: Informatics in Education – International Journal, Vol 7 No 1, p.55–74. (2008) Megtekinttés: 2021.08.27. https://hal.archives-ouvertes.fr/hal-00588766/document

83. E. BARENDSEN, N. GRGURINA, J. TOLBOOM: A New Informatics Curriculum for Secondary Education in The Netherlands. In: Brodnik A., Tort F. (eds) Informatics in Schools: Improvement of Informatics Knowledge and Perception. ISSEP 2016. LNCS, vol 9973. Springer, Cham. https://doi.org/10.1007/978-3-319-46747-4_9 Megtekintés: 2021.10.25. https://www.academia.edu/30476203/A_New_Informatics_Curriculum_for_Secondary_Education_in_The_Netherlands

84. Maciej SYSLO, Anna KWIATKOWSKA: Introducing a New Computer Science Curriculum for All School Levels in Poland. In: In: Brodnik A., Vahrenhold J. (eds) Informatics in Schools. Curricula, Competences, and Competitions. ISSEP 2015. LNCS, vol 9378. Springer, Cham. https://doi.org/10.1007/978-3-319-25396-1_13. Megtekintés: 2021.10.25. https://www.researchgate.net/publication/300253595_Introducing_a_New_Computer_Science_Curriculum_for_All_School_Levels_in_Poland

85. P. HUBWIESER, M. BERGES, J. MAGENHEIM, N. SCHAPER, K. BRÖKER, M. MARGARITIS, S. SCHUBERT, L. OHRNDORF, BERDES: Pedagogical content knowledge for computer science in German teacher education curricula. In: ACM, WiPSE '13 Proceedings of the 8th Workshop in Primary and Secondary Computing Education, (2013). Megtekintés: 2021.08.29. https://www.edu.tum.de/fileadmin/tuedz01/ddi/Publikationen/2013/Hubwieser-et-al-WiPSCE-2013-Preprint.pdf

86. Norbert BREIER, Peter HUBWIESER: An Information-Oriented Approach to Informatical Education. In: Informatics in Education, Institute of Mathematics and Informatics, Vilnius vol.1 No:1 p:31-42. (2002) DOI:10.5555/827120.827124 Megtekintés: 2021.10.26 https://www.researchgate.net/publication/229041318_An_information-oriented_approach_to_informatical_education

87. Sellye János Gimnázium, Komárom Megtekintés: 2021.10.24. http://selye.gartproject.com/hu/44/az_iskola_pedagogiai_programja.html

88. Tamási Áron Líceum, Székelyudvarhely, „Tantervek," Megtekintés: 2021.10.24.: http://www.gimi.ro/tantervek

89. St. Mark's Sr. Secondary Public School, Meera Bagh, New Delhi, "Syllabus," Megtekintés: 2014.10.29.: http://www.saintmarksschool.com/meerabagh/academics/syllabus.html

90. Geek-<a/>-Hertz évenként megrendezett nemzetközi vetélkedő Megtekintés: 2021.10.24 https://geekahertz.in/

91. SM Sain Seri Puteri, Kuala Lumpur, „Subjek Ditawarkan, Sains Komputer" Megtekintés: 2014.10.29. http://seseri.edu.my

92. KERBER Zoltán: A középiskolai tanóra: kereszttantervi kompetenciák. Oktatáskutató és Fejlesztő Intézet (2009); Megtekintve: 2021.08.26. http://ofi.hu/kozepiskolai-tanora-kereszttantervi-kompetenciak

93. KÖRÖSNÉ MIKIS Márta (projekt vezető): Szakmai ajánlás az informatikai kereszttantervi követelmények teljesítéséhez (CD). Informatika-Számítástechnika Tanárok Egyesülete, (2002); Megtekintve: 2021.08.26. Felhasználási útmutató: https://isze.hu/tananyag/szakmai-ajanlas/

94. BURGETTI Mónika (et al.): 101 ötlet innovatív tanároknak = 101 ideas for innovative teachers. Budapest, Magyarország: Jedlik Oktatási Stúdió (2005); ISBN: 9638700017. Megtekintve: 2021.18.26. http://www.jos.hu/Konyv/0013/index.html

95. Jelentés a Magyar Közoktatásról sorozat. ISSN: 1219-8692 (angol nyelven: ISSN: 1418-6756)
LANNERT Judit (szerk), HALÁSZ Gábor (szerk): Jelentés a Magyar Közoktatásról 1995. Országos Közoktatási Intézet; (1996);
HALÁSZ Gábor (szerk), LANNERT Judit (szerk), BALOGH Miklós (szerk): Jelentés a Magyar Közoktatásról 1997. Országos Közoktatási Intézet; (1998); ISBN: 9636824622 Megtekintve: 2021.08.26. http://ofi.hu/tudastar/jelentes-magyar/jelentes-magyar-090617
HALÁSZ Gábor (szerk), LANNERT Judit (szerk): Jelentés a magyar közoktatásról 2000. Országos Közoktatási Intézet; (2000); Megtekintve: 2021.08.26. http://mek.oszk.hu/08400/08451/08451.pdf
HALÁSZ Gábor (szerk), LANNERT Judit (szerk): Jelentés a magyar közoktatásról 2003. Országos Közoktatási Intézet; (2003); Megtekintve: 2021.08.26. http://mek.niif.hu/01300/01399/01399.pdf
HALÁSZ Gábor (szerk), LANNERT Judit (szerk): Jelentés a magyar közoktatásról 2006. Országos Közoktatási Intézet; (2006); Megtekintve: 2021.08.26. http://mek.oszk.hu/08400/08429/08429.pdf
BALÁZS Éva (szerk), KOCSIS Mihály (szerk), VÁGÓ Irén (szerk): Jelentés a magyar közoktatásról 2010. Oktatáskutató és Fejlesztő Intézet; (2011); Megtekintve: 2021.08.26. http://mek.oszk.hu/12800/12893/12893.pdf

96. Nemzeti Infokommunikációs Stratégia 2014–2020
Megtekintve: 2021.08.26. https://2010-2014.kormany.hu/download/b/fd/21000/Nemzeti%20Infokommunik%C3%A1ci%C3%B3s%20Strat%C3%A9gia%202014-2020.pdf

97. CSÉPE Valéria, Oktatás 2030 Tanulástudományi Kutatócsoport: A Nemzeti alaptanterv tervezete 2018. augusztus 31. p. 256–268 (2018). Megtekintve: 2021.18.26. https://www.oktatas2030.hu/wp-content/uploads/2018/08/a-nemzeti-alaptanterv-tervezete_2018.08.31.pdf

98. GEOMATECH 2015. Megtekintés: 2015.05.01. http://tananyag.geomatech.hu/
99. Marko PETKOVSEK, Herbert S. WILF, Doron ZEILBERGER: A=B (with foreword by Donald E. Knuth). Wellesley, MA, USA; A K Peters. Ltd.; (1996); ISBN: 1568810636. Megtekintés: 2021.08.26. http://www.math.upenn.edu/~wilf/AeqB.pdf
100. John DEWEY: A nevelés jellege és folyamata. ford.: Molnár Magda; Budapest, Tankönyvkiadó, (1976). ISBN 9631722201
101. William H. KILPATRICK: The Project Method - The Use of the Purposeful Act in the Educative Process. In: Teachers College Columbia University, New York, Vol. XIX, Nº 4. (1918) Szöveg publikálása: Derek Gillard (2018) Megtekintve: 2021.08.26. http://www.educationengland.org.uk/documents/kilpatrick1918/index.html
102. Michael KNOLL: The Project Method: Its Vocational Education Origin and International Development. In: Journal of Industrial Teacher Education Vol 34 No 3, p. 59–80 (1997). Megtekintve: 2021.08.26. https://scholar.lib.vt.edu/ejournals/JITE/v34n3/Knoll.html
103. John L. PECORE: From Kilpatrick's Project Method to Project-Based Learning. In: M. Y. Eryaman & B. C. Bruce (eds.) International Handbook of Progressive Education, p: 155–171. (2015) Megtekintve: 2021.08.26. https://ir.uwf.edu/islandora/object/uwf%3A22741/datastream/PDF/view
104. M. NÁDAS Mária: A projektoktatás elmélete és gyakorlata. Magyar Tehetségsegítő Szervezetek Szövetsége, Magyarország, (2010)
105. HORVÁTH Attila (szerk): Újszerű pedagógiai módszerek bevezetése – vezetőknek, vezetőkről. Educatio Társadalmi Szolgáltató Nonprofit Kft, Budapest, (2011)
106. HORTOBÁGYI Katalin (szerk): Projekt kézikönyv. Text-Print Kft, Győr (2002) (az 1991-es azonos nevű kiadvány javított és bővített kiadása)
107. KNAUSZ Imre: A tanítás mestersége. Egyetemi jegyzet Miskolc, (2001). Megtekintve: 2021.08.26. http://www.mek.iif.hu/porta/szint/tarsad/pedagog/modszer/tanitas/tanitas.htm tömörítve pdf verzió: http://www.mek.iif.hu/porta/szint/tarsad/pedagog/modszer/tanitas/tanitasp.zip
108. HAVASS Miklós és LENGYEL Veronika (koordinálásával) Bakonyi P., Bálint L., Bendzsel M., Csaba L., Cser L., Detrekői Á., Dömölki B., Gábor A., Gordos G., Jávor A., Kárpáti A., Komenczi B., Lengyel V., Magyar G., Martos B., Sallai Gy., Szabó K., Szekfű A., Takátcs Gy., Tamás P., T. Bíró K., Vető I., Vonderviszt L.: Magyar válasz

az Információs Társadalom kihívásaira (szakértői vitaanyag); 1999. október 1. NJSZT-től elkérhető. Itt volt: http://itf.njszt.hu/23r4r23r/uploads/2013/08/MagyarValasz.pdf

109. Talyigás Judit (szerk): Tézisek az információs társadalomról. Miniszterelnöki Hivatal, Budapest (2000) ISBN: 1939284386 Megtekintés: 2021.10.27. https://itf.njszt.hu/wp-content/uploads/Tezisek-az-informacios-tarsadalomrol.pdf

110. Magyar Informatikai Charta. Inforum 2000. Megtekintés: 2021.08.27. http://www.arte-faktum.hu/kozgaz/mic.html

111. Miniszterelnöki Hivatal Informatikai Kormánybizottsága: Nemzeti Információs Társadalom Stratégia v 1.0 2001. május 17. itt volt: http://itf.njszt.hu/23r4r23r/uploads/2013/08/nits_kesz.doc In: Informatikai stratégiai anyagok http://itf.njszt.hu/informatikai-strategiak (Megtekintés: 2021.08.26. Lapsánszki András: Közigazgatási jog I. MeRSZ – Akadémiai Kiadó 3.5 https://mersz.hu/lapsanszky-kozigazgatasi-jog-i/ → http://www.artefaktum.hu/kozgaz/nits_kesz.doc)
MITS In: Magyar Közlöny 143. szám II. kötet Megtekintés: 2021.08.26. https://magyar-kozlony.hu/dokumentumok/59b8474753ddca3e026a7de472cfb02986887180/letoltes

112. DOS – Magyarország Digitális Oktatási Stratégiája (A Korm. előterjesztés melléklete) (2016.06.30.) Elfogadva: 1536/2016. (X. 13.) Korm. határozat. Megtekintve: 2021.08.26.
https://digitalisjoletprogram.hu/files/55/8c/558c2bb47626ccb966050debb69f600e.pdf

113. HÜLBER László (szerk): Digitális-alapú iskolafejlesztési módszert megalapozó háttértanulmány-kötet. készült az EFOP-3.1.2-16-2016-00001 kiemelt projekt felhíváshoz (2017) Megtekintés: 2021.08.27. https://drive.google.com/drive/folders/1f9wNZ_L7yUGz-zQnOFni6iI9_2WPK5vW

114. EACEA: Eurydice Brief – Digital Education at School in Europe. Publications Office at European Union, Luxembourg (2019); ISBN: 9789294840219, DOI: :10.2797/339457 Megtekintés: 2021.08.26. https://eacea.ec.europa.eu/national-policies/eurydice/sites/eurydice/files/eurydice_brief_digital_education_n.pdf

115. RACSKO Réka: Összehasonlító vizsgálatok a digitális átállás módszertani megalapozásáról. Phd-értekezés Eger (2016) DOI: 10.15773/EKF.2017.002 Megtekintés: 2021.08.26. http://disszertacio.uni-eszterhazy.hu/32/19/Racsko_Reka_ertekezes.pdf

116. HORVÁTH László, MISLEY Helga, HÜLBER László, PAPP-DANKA Adrienn, M. PINTÉR Tibor and DRINGÓ-HORVÁTH Ida: Tanárképzők digitális kompetenciájának mérése – a

DigCompEdu adaptálása a hazai felsőoktatási környezetre. In: Neveléstudomány 2020/2. ELTE PPK 2020.06 p. 5-25; ISSN: 2063-9546
DOI: 10.21549/NTNY.29.2020.2.1 Megtekintve: 2021.08.26.
http://nevelestudomany.elte.hu/downloads/2020/nevelestudomany_2020_2_5-25.pdf

117. Péter SZLÁVI, László ZSAKÓ: Informatics as a Particular Feld of Education. In: Teaching Mathematics and Computer Science Vol 3 No 2, p.283–294 (2005). Megtekintés: 2021.10.03. https://tmcs.math.unideb.hu/load_doc.php?p=74&t=doc

118. A. COHEN és B. HABERMAN, „CHAMSA: five languages citizens of an increasingly technological world should acquire," ACM Inroads, Vol 1 No 4, p. 54–57, (2010). Megtekintés: 2021.08.29.
https://www.researchgate.net/publication/229052077_CHAMSA_Five_languages_citizens_of_an_increasingly_technological_world_should_acquire

119. Informatics Europe & ACM Europe Working Group on Informatics Education: Informatics education: Europe cannot afford to miss the boat. (2013). Megtekintés: 2021.08.29. http://www.informatics-europe.org/images/documents/informatics-education-acm-ie.pdf

120. Simon Peyton JONES, Bill MITCHELL, Simon HUMPHREYS: Computing at school in the UK. (2013). Megtekintés: 2021.08.26. http://research.microsoft.com/en-us/um/people/simonpj/papers/cas/computingatschoolcacm.pdf

121. SZALAYNÉ TAHY Zsuzsanna: Mi van, ha nem tudok teát főzni? In: Dr. Szlávi Péter, Dr. Zsakó László (szerk): INFODIDACT 2013, Informatika Szakmódszertani Konferencia; Zamárdi, Magyarország: Webdidaktika Alapítvány, (2013); Paper: TZs.pdf, 9 p. ISBN 9789630883870. Megtekintve: 2021.08.26.
https://people.inf.elte.hu/szlavi/InfoDidact13/Manuscripts/TZs.pdf

122. ECDL Foundation: Perception & Reality – Measuring Digital Skills Gaps in Europe, India and Singapore. ECDL Foundation (2018). Megtekintve: 2021.08.26.
https://icdleurope.org/policy-and-publications/perception-reality-measuring-digital-skills-gaps-in-europe-india-and-singapore/

123. J. M. WING: Computational thinking and thinking about computing. In: Philosoohical Transactions of The Royal Society A, Vol 366 No 1881, p. 3717–3725, (2008). Megtekintés: 2021.08.29.
https://www.researchgate.net/publication/23142610_Computational_thinking_and_thinking_about_computing

124. Péter SZLÁVI, László ZSAKÓ: Programming versus application. In: Mittermeir R.T. (szerk.) Informatics Education – The Bridge between Using and Understanding Computers. ISSEP 2006. LNCS, Vol 4226. p. 48–58. Springer, Berlin, Heidelberg. (2006) ISBN: 9783540482185 DOI: 10.1007/11915355_5. Megtekintés: 2021.08.29. https://www.academia.edu/5747262/Programming_Versus_Application

125. M M SYSŁO, A B KWIATKOWSKA: Informatics for All High School Students – A Computational Thinking Approach. In: Diethelm I., Mittermeir R.T. (szerk): ISSEP 2013. LNCS, Vol 7780 p. 43–56, Springer, Berlin, Heidelberg. DOI: 10.1007/978-3-642-36617-8_4. (2014) Megtekintés: 2021.08.29. https://www.researchgate.net/publication/262168697_Informatics_for_all_high_school_students_A_computational_thinking_approach

126. Danijel RADOSEVIC, Tihomir OREHOVAČKI, Alen LOVRENCIC: New Approaches and Tools in Teaching Programming. In: 20th Central European Conference on Information and Intelligent Systems (CECIIS), Varaždin, Croatia, (2009). Megtekintés: 202108.27. https://www.researchgate.net/publication/224930648_New_Approaches_and_Tools_in_Teaching_Programming

127. SZALAYNÉ TAHY Zsuzsanna (szerk): Beszámoló az Élenjáró Gimnáziumok Igazgatóinak Grémiuma (ÉGIG) szervezésében megtartott Informatika és technika munkaközösségi találkozóról Budapest, Békásmegyeri Veres Péter Gimnázium 2018.01.10. (2018) Megtekintés: 2021.08.26. http://egig.hu/tantargyak/informatika.pdf

128. TÓSZEGI Zsuzsanna: A képi információ. Országos Széchenyin Könyvtár, Budapest (1994) ISBN: 9632003373 Megtekintés: 2021.08.22. https://mek.oszk.hu/03100/03123/

129. BALLA Fanni: Az auditív és a vizuális ingerek hatása a szövegértésre. In: Anyanyelvpedagógia Vol 12 No 3, (2019) p: 31–37. ISSN: 2060-0623; DOI: 10.21030/anyp.2019.3.3; Megtekintés: 2021.08.21. http://www.anyanyelv-pedagogia.hu/img/keptar/2019_3/Anyp_XII_2019_3_3.pdf

130. Zsuzsanna SZALAYNÉ TAHY: Teaching Programming Indirectly with “Paint”. In: Matevž JEKOVEC (eds): ISSEP 2015. University of Ljubljana, Faculty of Computer and Information Science, Ljubljana (2015) p. 67., ISBN 9789616209878. Megtekintés: 2021.08.26. http://issep15.fri.uni-lj.si/files/issep2015-proceedings.pdf

131. PAPP Petra, CSERNOCH Mária: A táblázatkezelés is problémamegoldás? In: Szlávi Péter, Zsakó László (szerk): INFODIDACT’2018 p. 187–202. Webdidaktika Alapítvány (2019) ISBN 978-615-80608-2-0 Megtekintés: 2021.08.21. teljes:

https://people.inf.elte.hu/szlavi/InfoDidact18/Infodidact2018.pdf kézirat: https://people.inf.elte.hu/szlavi/InfoDidact18/Manuscripts/PPCsM.pdf

132. P. HUBWISER: Functional Modelling in Secondary Schools Using Spreadsheets. Education and Information Technologies,Vol 9 No 2, p. 175–183, (2004). Megtekintés: 2021.08.29. https://www.edu.tum.de/fileadmin/tuedz01/ddi/Publikationen/2000-04/2004-Hubwieser-IFIP-EaIT.pdf

133. Zsuzsanna SZALAYNÉ TAHY: How To Teach Indirectly – Using Spreadsheet Application. In: Acta Didacta Napocensia, Cluj-Napoca, Romania, ISSN: 2065-1430 Vol 9 No 1, p:15–22 (2016) Megtekintés: 2021.08.26. http://adn.teaching.ro/article_9_1_2.pdf

134. Bennett KANKUZI, Bassey ISONG, Lucia LETLONKANE: Using the Spreadsheet Paradigm to Introduce Fundamental Concepts of Programming to Novices. In: Southern Africa Computer Lecturers' Association 2017 conference, SACLA Magaliesburg, South Africa, p: 39–45 (2017). Megtekintés: 2021.08.26. https://www.researchgate.net/publication/318216386_Using_the_Spreadsheet_Paradigm_to_Introduce_Fundamental_Concepts_of_Programming_to_Novices

135. CSERNOCH Mária, BIRÓ Piroska: Sprego helye az informatika tantervekben. In: INFODIDACT 2015. Webdidaktika Alapítvány, p. 1-13 (#12) (2015) ISBN: 9789631238921. Megtekintés: 2021.08.26. https://people.inf.elte.hu/szlavi/InfoDidact15/Manuscripts/CsMBP.pdf

136. GREGORITS T.: *Force of Summation* In: Teaching Mathematics and Computer Science, Debrecen, 12/2 (2014), 185-199 DOI: 10.5485/TMCS.2014.0365 Megtekintés: 2021.08.26. http://tmcs.math.unideb.hu/load_doc.php?p=273&t=doc

137. Zsuzsanna SZALAYNÉ TAHY: Guess the code of conditional summation. In: Proceedings of the 10th International Conference on Applied Informatics Eger, Hungary p. 279–284; (2017); DOI: 10.14794/ICAI.10.2017.279. Megtekintés: 2021.08.26. https://icai.uni-eszterhazy.hu/icai2017/uploads/papers/2017/final/ICAI.10.2017.279.pdf

138. CSERNOCH Mária, BIRÓ Piroska: Button-up technikák hatékonyságának vizsgálata informatika szakos hallgatók táblázatkezelés-oktatásában. In: Kozma Tamás, Perjés István (eds): Új kutatások a neveléstudományokban 2012. ELTE Eötvös Kiadó, (2013) p. 369–392. ISSN: 2062090X; Megtekintés: 2021.08.26. http://www.eltereader.hu/media/2013/10/Uj_kutatasok_2012_READER.pdf#page=353

139. SZALAYNÉ TAHY Zsuzsa: Indirekt bizonyítás tanítása a középiskola második osztályában. Szakdolgozat ELTE TTK, Budapest (1987)

140. G. L. HERMAN, M. C. LOUI, L. KACZMARCZYK és C. ZILLES: Describing the What and Why of Students' Difficulties in Boolean Logic. In: ACM Transactions on Computing Education, Vol 12 No 1, p. 3.1-3.28, (2012). Megtekintés 2021.08.29. http://publish.illinois.edu/glherman/files/2016/03/2012-TOCE-Boolean-misconceptions.pdf

141. Valentina DAGIENE, Sue SENTANCE: It's Computational Thinking! Bebras Tasks in the Curriculum. In: ISSEP 2016; LNCS 9973; Springer International Publishing; p. 28-39. DOI: 10.1007/978-3-319-46747-4_3. Megtekintés: 2021.11.27 https://www.researchgate.net/publication/308499063_It's_Computational_Thinking_Bebras_Tasks_in_the_Curriculum

142. Ken ROBINSON, Lou ARONICA (fordította: Bányász Réka): Kreatív iskolák, HVG Kiadó ZRt, Budapest (2018) ISBN: 9789633045510, Megtekintés: 2021.08.26. kivonat: https://hvg.hu/kkv.businessmagazin/20190217_egy_jo_tanar_feladatai_robinson Eredeti: Creative Schools: The Grassroots Revolution That's Transforming Education; Megtekintés: 2021.08.26. https://pdfroom.com/books/creative-schools-the-grassroots-revolution-thats-transforming-education/j9ZdYz9rgV4/download

143. Sue SENTANCE, Andrew CSIZMADIA: Computing in the curriculum: Challenges and strategies from a teacher's perspective In: Andrej Brodnik & all (eds): Education and Information Technologies, Springer UK vol22, No.2, p. 469-495 (2017) ISSN: 1573-7608 DOI: 10.1007/s10639-016-9482-0 Megtekintés: 2021.08.26. https://link.springer.com/content/pdf/10.1007%2Fs10639-016-9482-0.pdf

144. Y. HOFOKU, S. CHO, T. NISHIDA és S. KANEMUNE, Why is programming difficult? – Proposal for learning programming in "small steps" and a prototype tool for detecting "gaps". In: Ira Diethelm (eds): Informatics in Schools. Local Proceedings of the 6th Conference ISSEP 2013 – Selected Papers p. 13–24., Oldenburg, Germany, Universitätsverlag Potsdam (2013) ISBN: 9783869562223 Megtekintés: 2021.08.26. https://books.google.hu/books?id=T86PubrNx-nsC&pg=PA13&lpg=PA13#v=onepage&q&f=false

145. F. A. DORÇA, L. V. LIMA, M. A. FERNANDEZ, C. R. LOPES: A Stochastic Approach for Automatic and Dynamic Modeling of Students' Learning Styles in Adaptive Educational Systems. In: Informatics in Education, Vol 11 No 2, p. 191-212, (2011). Megtekintés: 2021.08.29. https://www.researchgate.net/publication/285011173_A_

Stochastic_Approach_for_Automatic_and_Dynamic_Modeling_of_Students'_Learning_Styles_in_Adaptive_Educational_Systems

146. Zsuzsanna SZALAYNÉ TAHY, Zoltán CZIRKOS: Progress Log for Mentoring Programming Education. In: ISSEP 2017 p. 1–6, (2017); Megtekintés: 2021.08.26. http://issep2017.cs.helsinki.fi/files/short/progress-log-for-mentoring-programming-education.pdf

147. SZALAYNÉ TAHY Zsuzsanna, CZIRKOS Zoltán: Programozástanulás Mentorálása Haladási Naplóval. In: INFODIDACT 2017. Webdidaktika Alapítvány, p. 1–12 (#19) (2017) ISBN: 97896158060813 https://people.inf.elte.hu/szlavi/InfoDidact17/Manuscripts/SzTZsCZ.pdf

148. Zsuzsanna SZALAYNÉ TAHY, Zoltán CZIRKOS: The Two Worlds of Programming. In: Veronika Stoffová (eds): New Methods and Technologies in Education and Practice – XXXth DidMatTech; Trnava University in Trnava Faculty of Education, Trnava p. 59–67; (2017); ISBN: 9788056800294; Megtekintés: 2021.08.26. http://didmattech.truni.sk/2017/proceedings.pdf

149. Zsuzsanna PAPP-VARGA, Péter SZLÁVI, László ZSAKÓ: ICT teaching methods - Programming languages. In: Annales Mathematicae et Informaticae, No 35 p. 163–172 (2008) ISSN: 1787-5021 (Print) 1787-6117 Megtekintés: 2021.08.27. https://ami.uni-eszterhazy.hu/uploads/papers/finalpdf/AMI_35_from163to172.pdf

150. P. SZLÁVI és L. ZSAKÓ: Methods of teaching programming. In: Teaching Mathematics and Computer Science; Institute of Mathematics, University of Debrecen; Vol 1 No 2, p. 247–258., (2003). DOI: 10.5485/TMCS.2003.0023 ISSN: 2676-8364; Megtekintve: 2021.08.26. http://tmcs.math.unideb.hu/load_doc.php?p=15&t=doc

151. SZÖGI László: Az Eötvös Loránd Tudományegyetem története képekben. Eötvös Kiadó, ELTE Budapest, (2015). ISBN: 9789632847290; Megtekintve: 2021.08.26. https://www.elte.hu/file/ELTE_tortenete_SzogiL.pdf

152. A Műegyetem rövid története 1782–2000. In: KISS Márton (eds): Milleneumi Évkönyv. BME, Budapest (2000). ISBN: 9634206492, Megtekintés: 2021.08.26. https://www.bme.hu/sites/default/files/csatolmanyok/BME_1782-2000.pdf

153. KOLTAI Dénes, LADA László (szerk.): Az Andragógia korszerű eszközeiről és módszereiről. Tanulmánykötet, Nemzeti Felnőttképzési Intézet. Budapest, (2006); ISBN: 639649228; Megtekintés: 2021.08.26. http://www.elib.hu/06400/06496/06496.pdf

154. Stephen T FREZZA, Mats E DANIELS, Arnold N. PEARS, Åsa CAJANDER, Viggo KANN, Amanpreet KAPOOR, Roger Mc DERMOTT, Anne Kathrin PETERS, Mihaela SABIN, Charles Robert WALLACE: Modelling Competencies for Computing Education beyond 2020: A Research Based Approach to Defining Competencies in the Computing Disciplines. In: Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '18 Companion), July 2–4, 2018, Larnaca, Cyprus. ACM, New York, NY, USA; p 148–174. (2018) DOI: 10.1145/3293881.3295782, ISBN: 9781450362238; Metgekintés: 2021.08.26. https://dl.acm.org/doi/pdf/10.1145/3293881.3295782

155. Creating Exams. In: Teaching Excellence & Educational Innovation, Carnegie Mellon University Eberly Center. (2020); Megtekintés: 2021.08.26. https://www.cmu.edu/teaching/assessment/assesslearning/creatingexams.html

156. Christoph STADTFELD, András VÖRÖS, Timon ELMER, Zsófia BODA, Isabel J. RAABE: Integration in emerging social networks explains academic failure and success. In: Proceedings of the National Academy of Sciences, National Academy of Sciences ISSN: 0027-8424, vol 116, No: 3, p: 792-797, (2019), DOI: 10.1073/pnas.1811388115, https://www.pnas.org/content/116/3/792.full.pdf

157. Robert C. MARTIN: The Clean Coder: A Code of Conduct for Professional Programmers ISBN: 9780137081073 / Rézműves László ford: Túlélőkönyv programozóknak: Hogyan váljunk igazi szakemberré? Kiskapu Kiadó Budapest (2011) ISBN: 9789639637863

158. Andrew TROTMAN: Why don't European girls like science or technology. Microsoft Stories Europe (2017) Megtekintés: 2021.08.26. https://news.microsoft.com/europe/features/dont-european-girls-like-science-technology/

159. Zsuzsanna SZALAYNÉ TAHY: Guess the Code. In: The 9th International Conference on Informatics in Schools, 2016.10.13-15. Münster, Germany (2016). Megtekintés: 2021.08.26. http://issep2016.ens-cachan.fr/WorkShops/ISSEP16_WS_GuessTheCode.pdf

160. SZALAYNÉ TAHY Zsuzsanna: Guess the Code! In: Magyar Science On Stage Fesztivál 2018.10.05-07. Szeged, Agóra. (2018) Megtekintés: 2021.08.26. http://szinpadon-a-tudomanyhu.web-server.hu/2018/SzalayneTahyZsuzsa_long.pdf