AALTO UNIVERSITY
School of Science
Department of Computer Science and Engineering

Sampo Kivirinta

Reducing Persisting Cognitive Dissonance and Drop-out Rates in Computer Science 1 Using Visual Debugger Aid

Master's Thesis
Espoo, December 16th, 2014

Supervisor: Professor Lauri Malmi
Instructor: Juha Sorva D.Sc. (Tech.)

AALTO UNIVERSITY
School of Science
Department of Computer Science and Engineering

ABSTRACT OF
MASTER'S THESIS

Author:	Sampo Kivirinta		
Title:	Reducing Persisting Cognitive Dissonance and Computer Science 1 Drop-out Rates Using Visual Debugger Aid		
Date:	Dec 16, 2014	**Pages:**	76
Professorship:	Software Technology	**Code:**	T-106
Supervisor:	Professor Lauri Malmi		
Instructor:	Juha Sorva D.Sc. (Tech.)		

Constructivism provides an excellent learning model but its methods are associated with high cognitive load, which is further increased by the inherent cognitive complexity of introductory computing courses (CS1). On the other hand, the introductory nature of CS1 means high variation in existing skills. This variation combined with external limitations and poor metacognitive skills can lead to unresolved conflicts and persisting cognitive dissonance. Persisting cognitive dissonance harmfully interacts with high cognitive load by further taxing working memory, preventing conflict resolution and leading into a vicious circle of accumulating cognitive dissonance. This work identifies three harmful interations through which persisting dissonance damages learning in CS1 and further escalates the dissonance: resource depletion, excessive cognitive load and avoidance failure. The resulting accumulation of dissonance is destructive and inevitably leads to avoidance behavior that can culminate in the abandonment of the studies.

Fortunately cognitive dissonance can be reduced by several techniques including positive framing and improving metacognitive skills. To demonstrate that many of these measures could be mediated by supporting scaffolds in the learning environment, this work proposes a prototype scaffold Eclipse DAPS (Debugging Assistant for Programming Students using Eclipse Python). DAPS provides assistance in debugging, one of the most difficult tasks a novice programmer must face. DAPS is a proof of concept intelligent tutoring system for teaching CS1 Python and aims to maintain student motivation by providing visualizations, metacognitive support and an integrated debugger. In addition, memorization techniques and easy repetition are used to improve knowledge encoding.

Keywords:	introductory programming, CS1, persisting cognitive dissonance, cognitive dissonance, failure rate, debugging, python
Language:	English

AALTO YLIOPISTO
Perustieteiden korkeakoulu
Tietotekniikan laitos

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä:	Sampo Kivirinta		
Työn nimi:	Persistoivan kognitiivisen dissonanssin ja ensimmäisen ohjelmointikurssin keskeytysmäärien vähentäminen käyttäen visuaalista debuggausassistenttia		
Päiväys:	16.12.2014	**Sivumäärä:**	76
Professuuri:	Ohjelmistotekniikka	**Koodi:**	T-106
Valvoja:	Professori Lauri Malmi		
Ohjaaja:	TkT Juha Sorva		

Konstruktivismi tarjoaa erinomaisen oppimismallin, mutta sen metodit liitetään korkeaan kognitiiviseen taakkaan, jota lisää myös ohjelmoinnin alkeiskurssien korkea kognitiivinen monimutkaisuus. Toisaalta ohjelmoinnin alkeiskursseilla opiskelijoiden taidot vaihtelevat huomattavasti. Yhdistettynä sopeutumisrajoituksiin ja heikkoihin metakognitiivisiin taitoihin tämä variaatio voi johtaa sevittämättömiin konflikteihin ja siten myös persistoivaan kognitiiviseen dissonanssiin. Persistoiva kognitiivinen dissonanssi vuorovaikuttaa vahingollisesti kognitiivisen taakan kanssa kuluttamalla työmuistia ja estäen konfliktien ratkaisua ja johtaa lisääntyvän dissonanssin noidankehään. Tämä työ tunnistaa kolme vahingollista vuorovaikutusta, joiden kautta persistoiva dissonanssi heikentää aloittelevien ohjelmoijien oppimista ja edelleen lisää heidän dissonanssiaan: resurssien loppuminen, liiallinen kognitiivinen taakka ja välttämisen epäonnistuminen. Lopputuloksena syntyvä dissonanssin kertyminen on tuhoisaa ja johtaa väistämättä välttämiskäyttäytymiseen, joka voi huipentua kurssin keskeyttämiseen.

Onneksi kognitiivista dissonanssia voidaan vähentää useilla tekniikoilla mukaan lukien positiivinen kehystäminen ja metakognitiivisten taitojen kehittäminen. DAPS (Debugging Assistant for Programming Students, debuggausassistentti ohjelmoinnin opiskelijoille) on älykkään opastusjärjestelmän prototyyppi, joka pyrkii tarjoamaan demonstraation kyseisten tekniikoiden ohjelmallisesta implementaatiosta. DAPS auttaa aloittelevia ohjelmoijia Python-kielisten ohjelmien debuggauksessa ja ylläpitää motivaatiota visualisaatioilla ja metakognitiivisella tuella. Lisäksi muistinnustekniikoita ja helppoa toistoa käytetään tehostamaan informaation muistiin koodamista.

Avainsanat:	ohjelmoinnin opettaminen, ensimmäinen ohjelmointikurssi, persistoiva kognitiivinen dissonanssi, keskeyttäminen, debuggaus, Python
Kieli:	englanti

Acknowledgments

I would like to thank Lauri Malmi and Juha Sorva for their exemplary guidance. I would also like to thank Lauri Kantola for several in-depth discussions in the field of cognitive psychology.

In addition many other people within and outside Aalto University assisted in making this work, including several members from LeTech[1]. Even when their input was lesser in quantity, it still remains an equally crucial part of the whole process. I am deeply grateful to you all.

[1] Learning + Technology Group, Department of Computer Science and Engineering, School of Science, Aalto University

Table of Contents

List of Figures and Tables

Before You Begin

It is strongly recommended to glance through the table of contents before reading further. The titles are specifically written to provide a good overview of the contents by stating firstly the domain of discussion and secondly the most important conclusion.

The three main parts of the work are chapters 2-4:

2. Literature review of cognitive psychology that investigates motivational etiology behind CS1 drop-out rates and ways of alleviating these problems.
3. The formulation of *the persisting dissonance behind CS1 drop-out rates* (the leading finding in this work) and the translation of found theories into design requirements for Eclipse DAPS.
4. Specification and implementation of Eclipse DAPS.

1 Introduction

Computing permeates our society like never before and processes are increasingly implemented through programmed software. This has created a great need for efficient education to create the needed programming professionals. One severe problem in the education of programming are the drop-out rates in the elementary programming courses (Computer Science 1, CS1). While the existence of the problem is easily verified by course personnel, the solutions have been hard to find.

This thesis investigates the problem through cognitive psychology by concentrating on its motivational foundations, identifying what support systems the student's motivational system would need in order to persevere through CS1 and finally begins implementing a programming assistant system that solidifies these theoretical findings into a functional tool – Eclipse DAPS.

1.1 Cognitive Psychology – The Solid Level Below

The thesis leans heavily towards cognitive psychology (or cognitive science) in its efforts to help students in their difficult task of learning programming. One might ask, why cognitive psychology? There certainly is a wast number established teaching and learning disciplines. Still, while cognitive psychology is not a guidebook to teaching, the subjects it investigates are of a sufficiently "low level" for being isolated and investigated with quantitative methods yielding consistent and hard evidence. It is a good and sturdy foundation for building.

Cognitive psychology is the study of higher mental processes (Gerrig & Zimbardo 2002). It investigates matters such as:

- attention (the selection of relevant over irrelevant)
- memory (the storage and timely retrieval of relevant information)
- perception (reception and interpretation of external stimuli)
- language
- metacognition (self-knowledge and self-regulation, planning, monitoring, evaluating).

While these processes are far from practicalities of teaching, they are extremely relevant to the subject at hand. It is these processes that must be guided and nurtured to lead the students to successful learning and understanding. The understanding of these mental processes, their capabilities and their limitations, is therefore, important.

While from the perspective of cognitive psychology, mental processes such as attention and memory are called higher mental processes, in some instances they can be referred as basic or simple mental processes due to the different perspective. If we consider the information layered as a hierarchy, this thesis considers these processes as the level below its focus and aims to identify structures directly over them. The used terminology is, as natural languages always are, context dependent.

1.2 Goals and Contribution of the Thesis

- This work approaches CS1 teaching from the cognitive psychology's perspective, reviews relevant theories, and investigates the motivational foundations behind CS1 drop-out rates.

- A new perspective of *persisting dissonance behind CS1 drop-out rates* is proposed, where *resource depletion*, *excessive cognitive load* and *avoidance failure* are identified as important components leading into a vicious circle of accumulating cognitive dissonance. This high persisting dissonance degrades student's performance and eventually enforces the student to abandon CS1 studies.

- Eclipse DAPS (Eclipse Debugging Assistant for Programming Students) is proposed. DAPS is an automated tutoring system prototype that uses cognitive techniques to teach CS1 programming in Eclipse Python environment. It aims to improve the student's motivation with visualizations, encouragement and instruction in metacognitive skills and attempts to protect the student from harmful repetition and deadlocks. Memory theories are employed to ease memorization, improve recall and reduce cue-dependent forgetting.

- DAPS design aims to create a basis for building a genuine resource for the curriculum and to address the generally poor adopting of these tools through creating positive experiences, and ease of use. DAPS does not fix problems but assists in their fixing and encourages learning, provides zero delay feedback and can be extended to information collection.

- The tool uses Eclipse IDE to reduce code base and maintenance costs, to ease future development and to allow easy tapping into Eclipse ecosystem's constantly developing resources.

1.3 What Is DAPS – Protection from Persisting Cognitive Dissonance

DAPS is an intelligent tutoring system (ITS) and has the following features and qualities:

- Main approach is to try to protect a CS1 student from circumstances that cause *persisting cognitive dissonance* and to help the students in resolving it. It does this by providing direction and metacognitive instruction.
- Main interaction in the form of approachable and familiar textual dialog that never leaves the student without direction.
- *Added value features* (debugger and visualizations) are used to create a productive interaction with the user. This motivates into using DAPS and provides the interaction into which the motivational and dissonance relieving elements can be incorporated.

2 Review of the Literature

2.1 Sociological Perspective – Teaching Does Not Benefit All Students Equally

The causality between teaching and learning is accepted with little criticism. Yet Coleman Report 1966 (Coleman 1968), which is called as *"the most important contribution by sociologists to research on schooling"* (Secada & Marrett 2000 p. 37), gives very little reason to trust this hypothesis. Its proposal of changing educational context by moving students between schools to improve learning results in weaker areas and reduce dispersion in education results, tells us how little the perceived quality of teaching correlated to the learning results obtained. When learning results were compared, the differences in teaching between the schools where far inferior to the differences in learning between the students. In the study *pupil attitude factor* had stronger impact on achievement than all school related factors combined. Moreover, only 10-20% of the achievement was associated with school differences.

The conflict is obvious. In one hand, teaching greatly improves learning results, one the other, poor learning results can only be marginally improved through better teaching. This leads us to ask a valid question: **Are there factors that inhibit some students from benefiting from the teaching?**

2.2 Motivation – Why Do They Quit?

Our society is ever more perforated by computing, but still CS1 students quit and turn away from this promising future and drop-out rates of 20-40 percents are reported by many institutes (Kinnunen & Malmi 2006) and a recent meta-analysis narrows this into a mean worldwide failure rate of 33,3% (Watson & Li 2014). Why do they quit? To answer this question we must investigate the system which is used in making these decisions – *the motivation*.

> *Motivation: The process of starting, directing, and maintaining physical and psychological activities; includes mechanisms involved in preferences for one activity over another and the vigor and persistence of responses.* (Gerrig & Zimbardo 2002)

The concept of *motivation* is used to answers following three questions (Pervin 2003 p. 105):

1. *What activates the organism?*
2. *Why does the organism select or choose one response over another, one direction of activity over another?*
3. *Given the same stimulus, why does the organism sometimes respond one way and sometimes another?*

While defining motivation gives us little insight into understanding it, the definition does provide a crucial starting point to understanding. **Motivation is an internal system of the person.**

But to maintain our own motivation in constructing a more general answer, let us consider one "partial" solution: *a subgoal*.

2.2.1 Some Important Terms

2.2.1.1 Goals – Should Always Be Reached

> *In social cognitive theory, desired future events that motivate the person over extended periods of time and enable the person to go beyond momentary influences.* (Pervin 2003 p. 101) The definition for a goal.

While the motivational system is highly complex, some specific goals can be easily identified and used for our benefit. These easily identifiable goals are our "partial solution" to the problem of motivation. By providing good goals, that is, goals that are desirable, clearly reachable and reachable with reasonable effort, we can increase the person's motivation with each reached goal. This motivation can then be kept up by providing more goals that in time will also be reached. **Goals are not dreams. Once set, they must be reached, or disappointment and loss of motivation will follow.**

2.2.1.2 Pull and Push

The several theories of *motivation* can be traditionally categorized into *push* and *pull* theories (Kelly 1958). *Push* theories work with motivational elements that compel the organism into activity or decision, through negative impulses of fear, pain, anxiety etc. *Pull* theories in turn work with hedonistic elements that reward the organism with pleasure.

2.2.1.3 Intrinsic and Instrumental Motives – Instrumental Can Be Harmful

Motivational elements can also be categorized into *intrinsic* and *instrumental* (same as extrinsic). Intrinsic motives are directly involved with the task itself (for example a desire to become better in the task or learn from it). Instrumental motives are related to the outcome of the task (for example getting paid for the end result, getting into even better task afterward, or getting a good grade). Of these motives, intrinsic correlate strongly with performance and persistence in the task, but quite surprisingly even when instrumental motives can lead to performing the task, instrumental motives reduce the positive effects of intrinsic motives (Wrzesniewski et al. 2014). To prevent the harmful effects of instrumental motives Wrzesniewski et al. recommend three organizational features to promote intrinsic motives:

1. *Small but regular reminders of organizational purpose can keep internal motives dominant and improve employee performance at the same time.*
2. Promote benefits such as skill mastery or impact to others, instead of high salary.
3. Avoid rigid supervision, explain tasks well and allow autonomy and individual approach.

2.2.2 Behaviorism

Behaviorism is: *a scientific approach that limits the study of psychology to measurable or observable behavior* (Gerrig & Zimbardo 2002). Behaviorism emerged to criticize the lack of empirical testing and data in previous research of psychology. The foundations of behaviorism are largely based on the work of Ivan Pavlov. In education, behaviorism is know by its widely used learning theory, in which the source of behavioral changes is the extensive repetition that is rapidly and constantly rewarded or discouraged based on the results.

2.2.2.1 Pavlov – Old Behavior Is Extremely Persistent

As the saying goes: Old habits die hard. The difficulty of unlearning previous behavior is evident. In the terms of behaviorism and its *classical conditioning* theory, this process of unlearning can also be called as *extinction*, a gradual reduction of the behavior's intensity and frequency after it has ceased to produce favorable outcome. However, extinction can not completely eradicate conditioning (Pavlov 1927). In Pavlov's view the extinction was not so much the weakening of the original behavior, but the formation of a new (and therefore much weaker) inhibitory behavior. In addition to slow extinction of the old unwanted behavior, the old behavior will rapidly recover (a process called *relearning*) if any positive feedback from it is perceived. This extreme persistence and dominance of older behavior is basic knowledge in psychology, but is often forgotten by the contemporary learning theories (Lamberts & Goldstone 2005).

2.2.2.2 Instrumental Learning – Perception of Fear Leads to Avoidance

Instrumental escape learning (Miller 1944) is a continuation of Clarke Hull's *Instrumental learning* model. Miller studied how conflicting motivations interact in *approach-avoid conflicts*. These conflicting motivations occur for example when the same goal (or the path leading to it) is both desired and feared, causing the resulting motivational and behavioral responses to act in opposite directions. A person who desires the goal, will try to approach it as long as the drive to approach exists. However the way to the goal can contain something that the person wants to avoid (for example due to fear). As the desire to approach the goal is stronger further away, the goal is approached. As avoidance steps in, the speed is gradually reduced until avoidance finally becomes dominant and completely stops the approach. If staying put at (or near) the threshold can not be maintained, the person will then retreat further away from the goal. While the behavior to approach and then give up by stopping or retreating may seem futile or illogical, this same behavior can be seen both in animals and humans. (Pervin 2003 pp. 110-113)

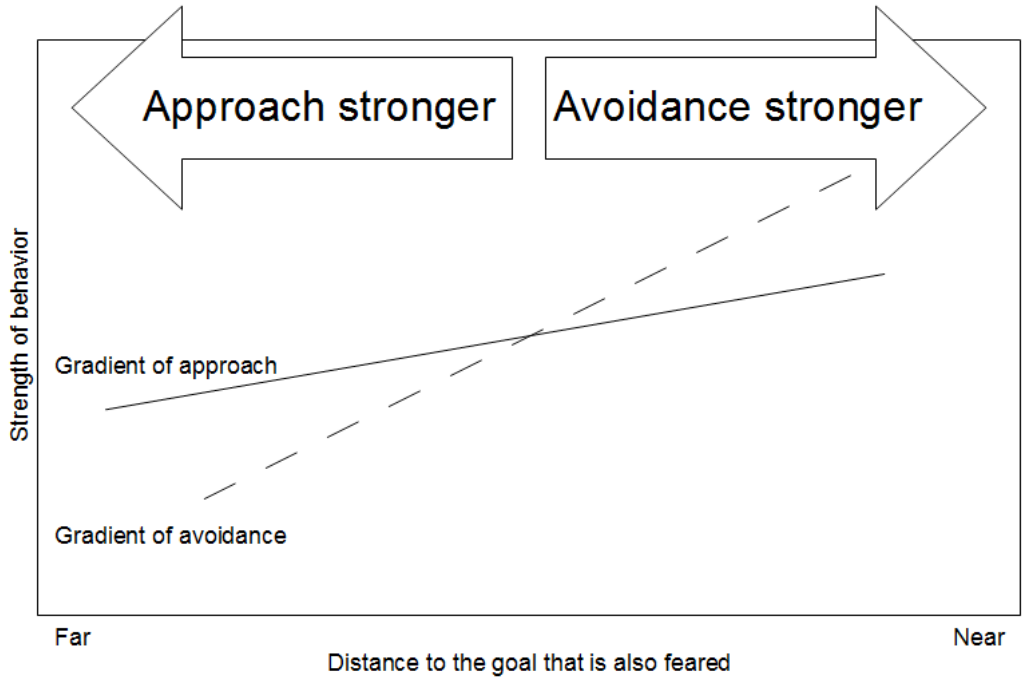

Illustration 1: Approach-avoidance conflict. Illustration based on (Dollard & Miller 1950 p. 356)

It is notable that the existence of problems does not prevent approaching the goal even when the goal is clearly unattainable. This leads into an alarming conclusion that the progress made in solving other difficulties, before being faced with the current problem, can not be used to reliably predict success in solving the current problem. In the context of CS1 students, there might be little or no warning, before approach-avoidance barrier is reached, and consequently even less time to resolve the difficulty, before conclusion to quit is reached.

As we will later see in this chapter. The major difficulty is that the problem has developed undetected and has already complicated into avoidance behavior. Its magnitude and complexity are unknown and it has remained hidden from the course personnel until the point where the student is powerless against it. It might even be that the student is already convinced at the impossibility of the problem, has already decided to quit and is actually seeking evidence in order to commit to the decision. In this case, the student would welcome hitting the barrier, as it would stop the progress and provide that last piece of evidence. This would allow the student to commit to quitting and to take the active steps needed. Clearly, at this point any intervention would be futile.

2.2.3 Cognitive Dissonance

2.2.3.1 Cognitive Dissonance – Removes Inconsistencies from Cognition

Cognitive dissonance is based on the theory of Leon Festinger and presented in his book *A theory of Cognitive Dissonance* (Festinger 1957). Festinger postulated that public statements in contrary to private beliefs represented *dissonant cognitions* that induce tension within that person and force the person to reduce the dissonance (Pervin 2003 p. 119). The theory is very well founded in the research of psychology and has even been directly tied to corresponding brain structures.

> *When our actions conflict with our prior attitudes, we often change our attitudes to be more consistent with our actions. This phenomenon, known as cognitive dissonance, is considered to be one of the most influential theories in psychology. However, the neural basis of this phenomenon is unknown. Using a Solomon four-group design, we scanned participants with functional MRI while they argued that the uncomfortable scanner environment was nevertheless a pleasant experience. We found that cognitive dissonance engaged the dorsal anterior cingulate cortex and anterior insula; furthermore, we found that the activation of these regions tightly predicted participants' subsequent attitude change. These effects were not observed in a control group.* (Veen et al. 2009) (in an article of Nature Neuroscience)

The theory is quite general and states that all *cognitions* (ideas, beliefs and knowledge about self and surrounding reality) and *behaviors* (actions) of the person should be in a *consonant* (consistent) state with each other. Any *dissonance* (inconsistency) between these cognitions will cause mental stress and discomfort to the person, forcing the person to find a new consonant state by changing or adding new cognitions or behaviors. The removal of the dissonance then allows the heightened stress and discomfort levels to return to normal.

The need for cognitive dissonance became from the apparent flaw in the reinforcement theories. According to them, greater rewards should strengthen the rewarded behavior.

However in several conditions people seemed to act in the exact opposite. For example persons receiving a large reward were less likely to change their views than persons receiving smaller rewards. According to the dissonance theory, receiving rewards could be used by the person to reduce the dissonance without accepting the differing view, where small reward would leave the person without justification to the discrepancy and force the person to adjust their private views to reduce the dissonance. (Pervin 2003 p. 119)

An another example would be the correlation of severe initiation to a group and loyalty to the group. Where *reinforcement* would consider the severe initiation a negative outcome and as a result would incorrectly predict it to inhibit the behavior (loyalty to the group). Cognitive dissonance states that hardship (initiation) without good results (the group being worth the endured effort) would be in dissonance with self-image. In other words, it would have been stupid to endure the hardship unless the membership was something worth the hardship. In order to not to have to devalue himself, the person needs to value the group's membership. (Pervin 2003 p. 119)

The second example gives reason to further alarm, because it shows how the persons actively change their perception to suit their needs and can even "falsify" this perception to escape acknowledging the conflict. Effectively, the biased views can escalate by further rewriting the perception of reality.

2.2.3.2 New Information Causes Cognitive Dissonance

Dissonance can be created by new information that is inconsistent with the person's cognitions. For an example, a plan to go to a picnic can be ruined by a weather report forecasting rain. If the inconsistency (the mutual exclusiveness of expecting good weather and high probability of bad weather) is recognized by the person, dissonance is created. (Festinger 1957 p. 4)

The severity of the created dissonance is a weighted proportion between the two dissonant clusters of cognitions and increases with:

- Degree with which the mutually relevant cognitions are in conflict (how much they exclude each other) (Festinger 1957 p. 13)
- Importance of the each individual cognition to the person (p. 16)
- Total number of dissonant cognitions (pp. 17-18).

Essentially all the different conflicting aspects of the new information are summed together.

> *The total amount of dissonance between this element and the remainder of the person's cognition will depend on the proportion of relevant elements that are dissonant with the one in question.* (Festinger 1957 p. 17)

The previous statement is interesting also in the sense that it illustrates the formation of persistence by showing how already formed and accepted misconceptions become more resistant to change. As knowledge is built on top of older knowledge, the new cognitions will be tied to the faulty cognitions and later on will carry their weight against changing the underlying faulty ones. The longer the person can "fortify" any faulty piece of information, the more difficult it will be to replace with a correct one.

The source of the information is also important, as information is evaluated in the context of its source. If a positively evaluated source would give the person information that the person evaluates as negative, this might change the person's evaluation of the source towards negative or the evaluation of the information towards positive. In any case, the stronger the attitude (positive or negative) towards the information's source, the stronger the change in opinion. (Festinger 1957 p. 8)

Resource limitations favor teaching in groups and selection of subjects and pace is always a compromise between many variables. In addition, it is difficult to estimate how much difficulty a given person is having with the current topic (especially in groups), as the difficulties are much more strongly tied to the learner, than to the teaching. As a result, it is difficult to know when too much dissonance is created, and when this dissonance starts to compromise the teacher's authority in the eyes of the weaker students.

2.2.3.3 A Person Actively Hides the Causes of Dissonance

Another highly problematic quality of the inconsistencies is their elusiveness. Festinger states that the person who has the inconsistency, will rarely, if ever, accept them as inconsistent, and will instead avoid admission through the use of rationalizations (Festinger 1957 p. 2). These rationalizations pose a problem for investigating the matter of CS1 students quitting, as the students in question are shielding themselves from the too uncomfortable real reasons, and instead readily present valid but less uncomfortable reasons. For example, it is clearly more comfortable to state desiring something else, than to admit failure as the reason for quitting. However, in these conditions, the "need for comfort" should be carefully considered. The "real" problem has already caused severe enough dissonance to cause the student to turn away from something that was previously (and possibly still is) considered valuable enough as a future career, a highly valuable goal. Quitting is a major failure for the student and as such a major source of additional dissonance, but this additional dissonance is forced on the student in the circumstances of already high dissonance that originally compelled the decision to quit. In these extreme conditions, strong avoidance behavior must exist to shield the student from the equally high mental stress and discomfort. Giving up this protection, by openly admitting to the real reasons, would be highly damaging and irrational.

As a result, the students would provide a much wider spectrum of reasons when questioned. Fittingly, in a qualitative questioning survey (Kinnunen & Malmi 2006) CS1 students presented a very wide array of reasons for dropping out. The writers concluded this variety as a major obstacle. Among other problem sources, they also identified low comfort level and accumulation of the dropping out reasons, both of which fit well to accumulating *cognitive dissonance*.

2.2.3.4 Dissonance Can Not Be Maintained

"The reality which impinges on a person will exert pressures in the direction of bringing the appropriate cognitive elements into correspondence with that reality." While this does not mean that all cognitive elements do not correspond to reality, it does mean that if they do not correspond, certain pressures must exist. (Festinger 1957 p. 11)

It is important to understand that the person is incapable of maintaining the dissonant state without a very high cost. If the new cognitions are to be accepted as true, they must be crafted consonant with the previous cognitions as soon as possible.

2.2.3.5 Reducing Dissonance – Requires Resources

> *The presence of dissonance leads to the action to reduce it just as, for example, the presence of hunger leads to action to reduce the hunger.* (Festinger 1957 p. 18)

Festinger provides three methods for reducing dissonance:

1. Change own behavior
2. Change environment to validate own behavior
3. Change own cognitions

Dissonance can be reduced by changes. If own behavior contributes to dissonance, it could be changed. For example learning that smoking is unhealthy, can be compensated by stopping smoking. The environment could be changed to give a reason to justify or necessitate own behavior, for example by participating to a smoking research, that requires participants to smoke. The third way of eliminating dissonance is through changing or adding cognitive elements. For example finding enough beneficial information, correct or incorrect, to reduce the value or importance of the harmful information (or its source) into negligible. (Festinger 1957 pp. 6, 19-22)

The third way of eliminating dissonance through changing cognitive elements can be troublesome. While incorrect information can be changed to correct, the reverse is also possible. The direction of these changes is that of the least amount of dissonance. In the circumstances of CS1 we clearly want the more correct information to be the less dissonant one.

2.2.3.6 The Cost of Reducing Dissonance Depends on the Person

The cost of reducing dissonance comes from the time and effort needed to process the new cognitions consonant with existing ones. As a person's resources are limited, resisting costly changes is clearly rational behavior and less costly options are naturally preferred.

However as the cost depends from the existing cognitions of the person, the cost of accepting a specific piece of new information varies greatly between persons. In externally controlled conditions (such as CS1) this variation of cost within finite resources creates an inevitable limit to the successful adoption of the given cognitions. If the cost of the adaptation exceeds the available resources, the information simply can not be accepted. The higher the cost, the wider the variation or the lower the resources available, the lower the percentage of the persons that can accept the new information. The rest must simply push it forward or ignore it as incorrect.

> *Making the change may simply not be possible. It would be a mistake to imagine that a person could consummate any change to his behavior if he wanted to badly enough.* (Festinger 1957 p. 26)

2.2.3.7 Persisting Dissonance Leads to Avoidance Behavior

> *When dissonance is present, in addition to trying to reduce it, the person will actively avoid situations and information which would likely increase the dissonance.* (Festinger 1957 p. 3)

Dissonance that can not be solved, is a continued burden on the person and forces ever poorer reduction methods to be employed. If we consider the context of CS1, avoidance is a very poor choice indeed. Reduction of exposure will directly translate into reduced opportunity to find solutions to the problem, in addition to hampering the studies in general. The problem is also intensified by the student's attempt to reduce the dissonance through searching information that belittles the problem or the subject in general.

When the process of limiting exposure to correct (dissonant) information and increasing exposure to "belittling" (but consonant) information progresses, the inconsistencies start to mount in a vicious circle, requiring even more resources to be expended on handling the escalating dissonance. Without rapid increase in student's resources or reduction of responsibilities, the dissonance will soon overcome all remaining motivational elements and enforce avoidance and escape form the unbearable circumstances.

> *The operation of a fear of dissonance may also lead to a reluctance to commit oneself behaviorally. There is a large class of actions that, once taken, are difficult to change. Hence, it is possible for dissonances to arise and to mount in intensity.* (Festinger 1957 p. 30)

2.2.3.8 Avoidance Is a Useful but Dangerous Strategy – Can Lead to Entrapment

If the problem can not be solved but can be fully avoided, all is well. In fact, if the problem never comes up again, avoidance can be a very good solution indeed, as the resources needed for the solution are saved. Even if one must ultimately face the problem, this type of approach might serve to push the resource expenditure until the needed resources are available. However, there is a danger, that the problem is perceived as solved, when it (and the cost of its solution) is only pushed forward.

Within CS1's highly interactive domain, avoidance becomes even more dangerous. In some cases avoidance of a specific (but severe) problem might incorrectly seem preferable, as general progress is still being made. Steady progress can provide additional confidence to push through and demonstrates ability to get past insignificant problems. However, steady progress can also mask the repeated failure in one specific problem and the subsequent development of ever stronger avoidance behavior towards that specific area. In addition to other harmful side-effects, this behavior can pile ever more resource costs into that specific problem, causing the problem to become completely unsolvable without a very large focused effort. Once developed, this aversion would manifest in an extraordinary strong dislike and clearly disproportionate difficulty to perform a problem associated task. The longer the "success through aversion"-strategy has prevailed, the more difficult it will be to break. In addition, as with any *dissonant cognition*, the person also avoids admitting the problem's existence or significance.

If the avoidance has been allowed to fully develop, it will similarly take much time and effort to diffuse. Even more importantly it will also require corresponding motivational resources,

which might be very hard to come by if others consider the matter as “should already understand”. For example, even when students know that they need help, asking or receiving it might not be socially acceptable. Enabling *help-seeking* (chapter 2.4.1.5) not only requires allowing help to be asked, but also providing the motivational resources for the work required. In any case, stronger avoidance reduces chances of success when the next *approach-avoid* barrier is confronted, each time forcing the student to recoil and to seek options for further avoidance, possibly even to drop out.

2.2.3.9 Focus to Process Instead of Results to Reduce Avoidance

The widely known phenomenon of *selective exposure to information* (preferring supporting over conflicting information) is especially strong in settings that involve sequential confrontation of new information. The resulting *confirmation bias* also becomes stronger over time as the commitment to the decision increases. However, this bias may be reduced by accountability in the decision making process instead of the accountability of decision outcome. (Jonas et al. 2001)

2.2.3.10 Connecting Dissonance, Motivation, Goals and Positive Reinforcement

While cognitive dissonance theory would seem to conflict with reinforcement, it can also be considered to deepen it. It adds the persistence of personality and self-image into the fold. The problem herein comes from not knowing what happens inside the black box that is the person. We can not exactly know which behavior the person is considering, and we can not exactly know how the outcome is evaluated.

Let us consider a previous example of the severe initiation leading to a higher loyalty towards the group. If the behavior considered would have been the decision to join the group and not the act of initiation itself, the behavior evaluated would be the person's own, and for the person's self evaluation, it would be favorable to be successful. If stated in these terms, the reinforcement theory also holds. It not only holds, but it dictates the result. The result must be good, because the results of own behavior must be desirable (or *dissonance* would be created). *The reinforcement* is working, but instead of being the directing force, it is now used for gaining the favorable outcome under the guiding persistence of *the planning* faculties of the person's *metacognition*.

In essence the person plays their *self-motivation* (trust in oneself) against the more immediate information perceived from their surroundings. The stronger the person's self-motivation is, the more discrepancy (dissonance) between the circumstances can be endured, before yielding to the outside constraints. However, the cost of persistence is the damage to the self-motivation in the case of being forced to yield.

This interpretation suits well to our aims in understanding the behavior of the students. If immediate positive feedback would be required and every hardship would instantly decimate their desire to complete their studies, there would be very few graduates. But even when the difficulties can be overcome with persistence of the metacognition, if sufficient reinforcement with negative result accumulates, this will diminish the value of the behavior to the point where even the metacognition's manipulation can not keep it viable. As the limit grows close, *the extinction* will set in, first leading to a half-hearted behavior (reduction in frequency and intensity), and ultimately to the abandonment of the behavior, admittance of failure, and the weakening of self-motivation.

> *Self-motivation is best sustained through a series of proximal subgoals that are hierarchically organized to ensure successive advances to superordinate goals... Pursuit of a formidable goal can sustain a high level of motivation provided it is broken down into subgoals that are challenging but clearly attainable through extra effort. To strive for unreachable goals is to drive oneself to unrelenting failure.* (Bandura 1989 p. 44)

> *How is it that we are able to pursue goals over extended periods of time, often while encountering periodic frustration? The pursuit of long-range goals involves the elements already noted as well as some additional elements. In particular, such longterm goal pursuit can be maintained through periodic accomplishment of subgoals and through self-reinforcement in terms of emotions such as pride and shame.* (Pervin 2003 p. 314)

This means that the division of goals into subgoals is not merely for the efficient execution of such tasks, but forms a vital source of reinforcement for the motivational system that provides the persistence to carry through. This is especially important within a curriculum or a course as the division of work into subgoals is done for the student, and can limit the student's capabilities for selecting reachable and sufficiently rewarding subgoals.

Moreover the resource limitations in catering to students with different backgrounds will inevitably leave some in poorer conditions as others. Previous incorrect or incompatible cognitions will either force them to ignore matters that will be needed in the future or work much harder than their classmates. Both of these options will drain their motivation, without comparable increase in positive reinforcement to support it. Even worse, the need to work harder than their peers can be incorrectly perceived as proof of inferiority and inevitable failure in studies.

2.2.4 Loss Aversion and Prospect Theory

> *The psychophysics of value induce risk aversion in the domain of gains and risk seeking in the domain of losses.* (Kahneman & Tversky 1984).

2.2.4.1 Perception of Risk Leads to Poor Performance

An another *behavioral economics* way of approaching motivation behind the CS1 drop-out rates could be through the *Prospect Theory* (Kahneman & Tversky 1979). According to the theory, people are inclined to avoid losses, even if that results in less gains. However, if there is no certain way of avoiding the loss, a dangerous behavior is taken, where avoiding loss is attempted by risking even greater loss. Risk seeking in *the domain of losses* (monetary, pain or human lives) is also widely confirmed by other researches. Unfortunately this risk taking leads into even higher losses and results in better performance in *the domain of gains* and poorer performance in *the domain of losses*. (Kahneman & Tversky 1984)

As taking risks in the domain of losses increases the losses even further and reduces the overall *value*, it is paramount that the CS1 students, who are exploring the new and unfamiliar circumstances which inherently contain much uncertainty, would consider themselves to act in the domain of gains. In essence they should always feel that there are ways which do not result in loss. This allows them to a retreat into safe and productive behaviors, if attempts of further gains should fail. The perception of being in the domain of losses would in turn

promote choices the students know to be bad, damaging or even immoral, but which could work with less work, for example favoring "quick and dirty" fixes over proper solutions. As persisting dissonance pushes the students in to the domain of losses, it will also promote these toxic behaviors.

2.2.4.2 Risk Taking Is a Serious Warning Sign

"*A large majority of people express a preference for the gamble over the sure loss."* (Kahneman & Tversky 1984). As no loss is preferred over a gamble and the gamble is preferred over a sure loss, the presence of gambling indicates the presence of the sure loss. In other words, the emergence of gambling tells us that the student considers the sure loss as a realistic outcome for the situation. Clearly this type of activity should not be present in any important tasks and should merit immediate intervention to prevent the imminent catastrophe. **Any task in CS1 that is associated with risk taking behavior should be considered for removal or change.**

2.2.4.3 Framing Effect – Positive Expressions Improve Motivation

The difference in behavior between domain of gains and domain of losses is made very interesting especially through *the framing effect* (*a cognitive bias*), which demonstrates how greatly a decision is affected based merely on its presentation. This framing effect can be used to manipulate *loss aversion* behavior, through altering the perception of gain or loss simply by stating the same decision positively of negatively (for example, of 600 hundred people, 200 will live, or 400 will die). Exactly as stated by *the prospect theory*, people will strongly favor the decision that does not contain the explicit negative outcome. Moreover, this effect is not affected by level of education nor from being confronted by the seeming illogicality of the choice. (Kahneman & Tversky 1984)

> *The bias that results from framing has been shown to be one of the most robust biases in human decision making.* (Thomas & Millar 2012)

While *framing* has its limits, it allows us to set the stage for learning in a positive way. In many circumstances *a domain of gains* can be created simply by using a suitable wording. For example an assignment might be be graded with 100 - errors (a domain of losses), or 50 points + bonus points (a domain or gains). In this example, a result of less than 50 points would still be deemed a failure, but the perception of results from 51-100 would be altered. Instead of avoiding penalty, the perception of superior accomplishment is created.

Framing research may also provide an another benefit as easier access to unbiased information reduces the framing effect (Thomas & Millar 2012). The erroneous negative perceptions can therefore be lessened simply by providing more unbiased information. As an example, if students incorrectly perceive some task too difficult to master, this faulty perception can be altered and the dissonance producing conflict solved by simply giving more information. Care should however be taken that the framing is not mixed up with "sugar coating" future difficulties, which could lead into entrapment by committing into unrealistic goals.

2.3 Memory

2.3.1 Cognitive Load

2.3.1.1 Working Memory – The Hard Limit for Complex Thought

One of the most founding limitations to cognitive processes is *the working memory capacity*. It limits the number of individual pieces of information that can be considered at any given moment, and while the exact number varies due to many different circumstances, its capacity is widely recognized to be about 7 (+/- 2) as stated by Miller (Miller 1994) in one of the most highly cited papers in psychology.

2.3.1.2 Chunks and Schemas – Circumvent Working Memory Capacity Limit

The concept of *chunks* (Chase & Simon 1973) aims to understand, how reasoning can still manage within working memory limitations against the near infinite variation of the real world. The concept is similar to using objects in Object-oriented programming (OOP), in which a previously constructed object template allows the relevant information to be automatically processed as one object without having to process each individual data field separately. Instead of many individual data items, only one chunk needs to be retained in the working memory. This effectively extends working memory capacity, but as with OOP the creation of these chunks requires previous work in constructing the object template.

Cognitive load theory uses *schemas* to describe the chunking process. To use our example in OOP, schemas are analogous to object temples. *"Schemas: General conceptual frameworks, or clusters of knowledge, regarding objects, people, and situations; knowledge packages that encode generalizations about the structure of the environment."* (Gerrig & Zimbardo 2002)

2.3.1.3 Cognitive Load – Is Caused by Element Interactivity

Cognitive load theory (Sweller 1988) continued by investigating the sources and management of under and over taxing the limited capacity of the working memory during learning. In short, high cognitive load prevents proper reasoning as working memory can not contain all of the required information elements, and required elements are constantly dropped to accommodate added elements.

The three types of cognitive load are:

- **Intrinsic** – Unavoidable complexity of the subject being learned
- **Extraneous** – Changeable conditions in which the learning happens
- **Germane** – Beneficial effort to create *schemas*

Intrinsic cognitive load is defined as constant and can not be manipulated, instead Sweller concentrated into finding effective management strategies and instruction designs for extraneous cognitive load.

> *It is pointed out that cognitive load theory deals with learning and problem solving difficulty that is artificial in that it can be manipulated by instructional design. Intrinsic cognitive load in contrast, is constant for a given area because it is a basic component of the material. Intrinsic cognitive load is characterized in terms of element interactivity. The elements of most schemas must be learned simultaneously*

> *because they interact and it is the interaction that is critical. If, as in some areas, interactions between many elements must be learned, then intrinsic cognitive load will be high. In contrast, in different areas, if elements can be learned successively rather than simultaneously because they do not interact, intrinsic cognitive load will be low. It is suggested that extraneous cognitive load that interferes with learning only is a problem under conditions of high cognitive load caused by high element interactivity. Under conditions of low element interactivity, re-designing instruction to reduce extraneous cognitive load may have no appreciable consequences. In addition, the concept of element interactivity can be used to explain not only why some material is difficult to learn but also, why it can be difficult to understand. Understanding becomes relevant when high element interactivity material with a naturally high cognitive load must be learned.* (Sweller 1994)

In CS1 there clearly is a high degree of interconnectivity. Not only in programming itself, but in supporting systems such as IDEs and compilers as well. This high *intrinsic cognitive load* necessitates good management of *extraneous cognitive load*.

2.3.1.4 High Cognitive Load Increases Damage from Dissonance and Avoidance

We must also consider the interactions of high *cognitive load* and accumulating *cognitive dissonance*. As we remember, dissonance that can not be handled immediately, leads to *avoidance* behavior. While avoidance in a domain with little to no interconnectivity could succeed fairly well, this does not hold for a domain of high interconnectivity. The avoided problem will resurface constantly and increase dissonance by drawing attention to the conflict and by increasing its importance. In addition, while understanding the avoided concept would provide additional help in learning the connected concept, it is instead damaging the learning. In the best case the new concept solves the avoided problem, but in the worst case the avoidance is extended to the connected concept. In both cases, the high level of interconnectivity enforces confrontation of conflicts and expenditure of resources to their resolution limiting the student's ability to manage their resource usage.

As a result, in this type of domain, the positive effects of avoidance (among its capability to reduce dissonance) are lost. In addition, the attempt can spawn other sources of dissonance and avoidance in an escalating fashion. This is even more unfortunate as we remember that avoidance is a defensive strategy. Further more, it is employed when other better strategies have already failed, and strong avoidance would therefore suggests absence of further fall back strategies. The severe weakening of this “last line of defense” is clearly dangerous. In CS1 circumstances, if the student is forced into avoidance behavior by whatever reason, the consequences are significantly worse, than in a domain of lower interconnectivity.

This conclusion gains support from experiments (Martinie et al. 2010) that found cognitive dissonance to increase performance in simple tasks, but damage performance on complex tasks involving working memory. They propose that dissonance interferes with working memory intensive tasks by expending working memory slots and thereby effectively reducing working memory size. This is further supported by Engle et al. who concluded that working memory is responsible for keeping mental representations active particularly in face of distraction (Engle et al. 1999).

One phenomenon that can explain this reduction in working memory capacity is that dissonance creates intrusive thoughts that are irrelevant to the task in hand but are related to the management or reduction of this discomfort. (Martinie et al. 2010 p. 11)

2.3.2 Levels of Processing Theory – Memorization Effectiveness

Memory retention is greatly affected by the way the information is processed at the time of learning. Semantic processing (e.g. popularity) of words produced 51-83% better recall compared to words with only non-semantic processing (e.g. whether a word fits a sentence) (Hyde & Jenkins 1973). Surprisingly, telling the subjects that their recall would be later tested (*intentional memorization*), provided only minor benefits compared to subjects who had not been preinformed (*incidental memorization*), and only if the words were associatively related to each other.

In more broader terms Craik & Lockhart proposed a framework named *Levels of Processing (Craik & Lockhart 1972)*. The key proposal is that the deeper the level of the processing is, the stronger the created memory traces are. A clear distinction is made between repetition by rote (*maintenance rehearsal*) that involves only repeating previous analysis and deeper more semantic analysis (*elaborative rehearsal*).

Eysenck & Keane sum up several other works that have refined the theory to investigate the type of beneficial elaboration. Single elaboration is superior to several. Simple is superior to complicated. Distinctive is superior to common. (Eysenck & Keane 2005 p. 207)[2]

2.3.3 "All memory Is cue-driven"

A cue (i.e. a recall cue) is any form of stimulus that promotes the recall of a memory. Successful application of any knowledge is dependent on its successful and timely retrieval from memory, and as stated by Lamberts & Goldstone: *"All memory is cue-driven"* (Lamberts & Goldstone 2005 p. 226). If the given context does not provide suitable cues for the student, then the relevant information is not remembered and will not be considered regardless of any other consideration. It is important to note that while there is a significant effectiveness difference between different cues in promoting the recalling specific memories, cue effectiveness also varies between students. If a problems provides a recall cue to to its solution, then only those who "get the cue" can benefit from it.

It is crucial to understand that cues are distinct from information. Cues are only memory devices needed for the functioning of the recollection process. A cue does not contain information, it only provides a reference through which the actual information can be accessed. Because of this, different rules apply to cues and information.

---

[2] The superiority of a single elaboration over several seems counter intuitive, as several examples clearly allow better understanding of the item itself. However the understanding of the item can not come before the item itself is identified with a handle (*the cue*) and the handle is anchored into memory with a crude description (*the elaboration*) to provide the location for storing the information. For this purpose, the elaboration should be single (one definition), simple (short) and distinctive (descriptions should not remind each other). This provides superior circumstances for memorization and recollection of the cue-elaboration pair, which is essential in preventing the associated information from becoming inaccessible and lost. Once the recollection of the cue is assured, the concept can be connected to other concepts and expanded through a multitude of views and examples that provide the actual understanding.

2.3.3.1 Cue-dependent Forgetting and Cue Overload – Limit the Number of Terms

The failure to recall is *forgetting*, which can be divided into *trace-dependent forgetting* (the information itself is lost) and *cue-dependent forgetting* (information is in memory, but inaccessible). Of these two *cue-dependent forgetting* seems dominant. The cues can be internal (such as mood) or external (such as smells, lighting or sounds). The cue's most important quality is the distinctiveness i.e. the cue's capability to *discriminate* between correct and incorrect responses. If the same cue maps to several items, its capability for successful recall is reduced. The same applies even when the mapping is unintentional i.e. the cue is mapped to an incorrect target. This phenomena of decreasing recollection effectiveness with increasing number of referred items, is called as *cue overload* (Watkins & Watkins 1975) (Lamberts & Goldstone 2005). While seemingly obvious, it is also important that the same cues are used in memorization and recall, as recall is lowered with any change in cues, even from weak cues to stronger ones. This means that if strong cues (that map to exactly one item) can be given to each item to be memorized, and these cues are again provided with recall, a very high recall rate can be achieved. (Eysenck & Keane 2005 pp. 220-228)

For our CS1 context, these findings provide significant insight. Cue distinctiveness, prevention of c*ue overload* and repetition of the used cues could all be combined into one technique: **Use a limited list of preselected key terms.** Other terms can be used, but their relation to the key terms must first be provided. This allows us to reduce the student's exposure to the plethora of "redundant" terminology that is present in modern day computing. For example terms namespace and scope are essentially different perspectives to the same concept. This redundancy is useful for a professional as the differing nuances can provide more accuracy and expressiveness in informed discussion. However, in the context of CS1, the benefits of these nuances are dwarfed by the new term inhibiting the student from recalling any useful information. If already familiar, albeit less accurate, term would have been used, the recall would have had significantly higher change of success, and the term dependent transfer of information could have been successful. The initial repetition should therefore use exactly the same cues, at least until the concepts become strong enough to be retrieved from memory also though other cues.

The rate of recall is reduced with time. At first the loss is very fast, but slows down as the time passes. Initially the memories are clear and fragile and slowly become faded and robust. This process can be explained as *consolidation* failures. To improve retention, the consolidation process can be enhanced by sleeping and sleeping between memorization and recall, improves recall results. The improvement is higher the earlier the sleep period occurs after memorization. (Eysenck & Keane 2005 pp. 225-228)

As stress damages sleep, damaging the *consolidation* of new memories is one of the ways with which persisting *cognitive dissonance* can inhibit learning.

2.3.3.2 CS1 Should Provide Good Cues – Information Comes Second

Cue-elaboration pairs form the foundation on which the information is built by forming the access points into the information and provide the superstructure through which the information is organized into the student's existing memory. If the multitude of examples is presented before the organizing structure of cue-elaboration pairs has been provided, the given information has no clear memory location to attach itself. It would remain loosely attached to various other memories and cues, each competing against the other in working

memory. In addition, the learner would have to choose the cues and elaborations by themselves. While this might provide unique perspectives in researching unexplored domain, in the case of thoroughly understood basics it would only lead into unnecessary confusion and wasted resources. In addition, any information presented before the selection of the cues is complete, would have to wait in limited working memory and would therefore have a very high chance of being lost before memorization can occur.

Cue-elaboration pairs can be provided with simple dictionary type listing of important concepts or more advanced techniques such as *mind maps* can be used to create or strengthen the organization of information and to provide visual aid in memorization of concept interactions. If the material is accessible, both of these techniques inhibit cue-dependent forgetting and prevent cues (and their associated information) from being lost.

A good example of a cue-elaboration pair would be: Ferrari is the king of cars. A bad example: Ferrari is an expensive red sports car and a men's cologne. The first is single, but the second example overloads the cue with additional meaning of cologne, which weakens the already weak new cue. “the king of cars” is short and simple (no lists), and while the red color is iconic, it can be added later. Lastly “the king of” is much more distinctive as “expensive”, “sports car” or “red”. Notice also that the concept of king shares the connotation of luxury, which is an example of even emotion eliciting deep processing that creates strong memory traces. The cue-elaboration pairs are only the pointers to the data, not the actual data records themselves. Their purpose is to simply facilitate memorization and recall, not to provide understanding of the subject.

Another example of using deep processing to connect abstract concepts to existing memories it to use a metaphor for physical world. This is one of the reasons for the wide success of object oriented programming. For example: A namespace is the home of the name. If oven is defined in the namespace, the home has it. If you use an oven, you would mean using the one in your current namespace. If you put something in the oven, you clearly must access it from the same namespace, as the food does not magically move to your friend's oven, just because you went there for a visit. The namespaces might also be nested (nest inside a nest) as you might have an oven in your nest (your shared student apartment) and a smaller oven in your nested own room. There is no magical connection between these ovens either, and you must remember which oven you are using, or terrible loss of food, overcooking or burnt food bugs will occur. Using the same namespace for everything is also doomed, as it would be impossible to remember which oven is whose with 5000 different people having heir 2000 identical ovens in the same huge apartment. Dividing people and ovens into smaller namespaces removes this burden.

2.3.3.3 Skill Learning and Repetition Priming – Repeated Cues Recalled Faster

> *Skill learning refers to the gradual improvement of performance with practice that generalizes to a range of stimuli within a domain of processing.* (Poldrack & Selco 1999 p. 208)

One example of repetition's capability to enhance *cues* that can be seen in *skill learning*, is *repetition priming*. This refers to the improvement that happens if a person is exposed to the same stimulus successively. With each successive exposure, the stimulus is processed faster and identified more accurately. Moreover functional MRI results suggest that *skill learning* and (both short- and long-term) *repetition priming* are neurally connected and are merely

different facets of the same learning system (Poldrack & Gabrieli 2001). This view is further supported by theoretical and computational analysis (Gupta & Cohen 2002).

2.4 Learning

2.4.1 Metacognition

The definition of metacognition is very diffuse and definitions vary especially between different fields of research. The used definition (Schraw 1998) is motivated by *educational psychology* and divides metacognition components into two main categories between knowledge and regulation:

Knowledge of cognition

Declarative knowledge: Knowledge of self as a learner and how factors affect own performance.

Procedural knowledge: How to do things (heuristics, strategies, categorizations)

Conditional knowledge: When and why to use declarative or procedural knowledge.

Regulation of cognition

Planning: Selection of strategy and resource management.

Monitoring: Maintaining awareness of current comprehension and progress.

Evaluation: Appraising results and task efficiency.

2.4.1.1 Metacognitive Instruction – Important In the Early Stages of Learning

Metacognition is a multidimensional, domain-general (improvements carry over into other domains at least partially) and can be improved through education. Significant improvement in learning can be gained by improving students' *metacognitive regulation* skills. Especially, metacognitive knowledge may compensate for low ability and lack of knowledge, which both are problems in the early stages of learning. (Schraw 1998)

2.4.1.2 Lack of Ability Does Not Affect Performance in Later Stages of Learning

The lack of knowledge is naturally removed in later stages of learning. The differences in performance caused by lack in ability (or IQ) converge during learning (Ackerman 1987). Swanson in addition found (Swanson 1990) that:"*Regardless of aptitude, higher metacognitive children performed better than lower metacognitive children.*"As the skill performance in later stages of learning will not depend on lack of knowledge and low ability will have no effect or only partial effect, the difficulties focus only on the initial stages of learning, not the end performance. This creates a compelling argument to compensate these early difficulties.

2.4.1.3 Metacognitive Instruction Improves Debugging (Problem-Solving)

If we consider the problem of drop-out rates in CS1, instruction in metacognitive knowledge should reduce the problems encountered during CS1 and through reduction of adversity, we should also see a comparable reduction in drop-out rates. Moreover, this reduction should come without reduction in end course skill level. If we in addition consider the difficulties in surviving from programming errors which are among the most difficult problem-solving circumstances encountered in CS1, Swanson makes a highly interesting claim (Swanson 1990):"*Thus, one may argue that measures of metacognition and general aptitude in the*

present study are tapping different forms of knowledge, and that high performance on the problem-solving tasks is more closely related to higher performance on the metacognitive measures than on the aptitude measures."

2.4.1.4 Metacognitive Skills – Can Be Improved by Intelligent Tutoring Systems

Roll et al. experimented with a system that provided "*immediate feedback on help seeking errors, general help-seeking instruction, and self-assessment episodes*" in geometry and concluded that the tutoring system can unintrusively assess a student's learning behavior, can improve the student's behavior in help-seeking, and the improvement in behavior extends to unassisted subjects (Roll et al. 2011).

2.4.1.5 Help-Seeking – Is Prevented by Fear of Losing Face

Help-seeking is an adaptive strategy used when encountered problems are perceived as too difficult to be solved independently. It is encouraged by a perception that help-seeking improves learning and hindered by fear of lower self-worth as a result of acknowledging the need for assistance and not being able to perform independently. (Roussel et al. 2011) They also concluded that goals such as *mastery-approach* ("learn as much as possible") and *friendship-approach* ("try to deepen relationships with my friends") correlate positively to *help-seeking* behavior which mediates achievement and social motivation. In addition goals such as *mastery-avoidance* ("try to avoid learning less than other students") and *friendship-avoidance* ("try to avoid conflict with friends") correlated inversely.

While the study was performed for high school students it should apply for CS1 as well. Also the similarity with *Prospect theory's* domains of *gains* and *losses* are evident, as with effects of *intrinsic* and *instrumental* motivations. Clearly there are many benefits in promotion of beneficial goals. It is also important to protect the students from harmful goals, such as maintaining face at the cost of not seeking help.

2.4.2 Bloom's Taxonomy

Bloom's taxonomy originates to *The Taxonomy of Educational Objectives, The Classification of Educational Goals, Handbook I: Cognitive Domain* (Bloom et al. 1956) and remains to be highly influential in the domain of teaching. A more recent version of the taxonomy (Anderson & Krathwohl 2001) is called here as *revised Bloom's taxonomy* following the nomenclature used by Sorva (Sorva 2012).

The original *Bloom's taxonomy* was one-dimensional and contained only *cognitive dimension*. This represents the increasing cognitive complexity of the tasks (Anderson & Krathwohl 2001). The revision added a second dimension, *the knowledge dimension*, which represents the increasing abstraction of the knowledge. In addition to the taxonomy itself Anderson & Krathwohl give detailed instructions in how to apply taxonomy into curriculum design.

Greatly simplified instructions would be along these lines:

1. Identify Education Objectives
2. Categorize Objectives according to the taxonomy
3. Choose teaching methods that align well with the objectives
4. Design tests that test chosen objectives with depth and/or breadth.

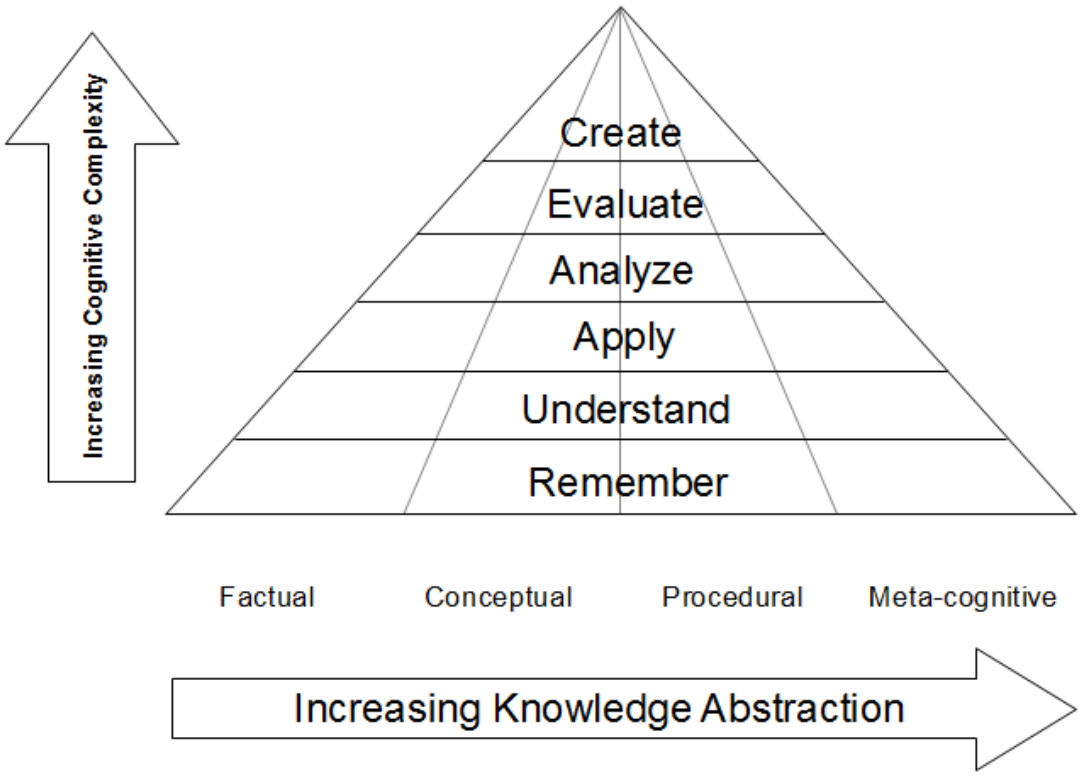

Illustration 2: The category names of cognitive dimension (remember - create) and knowledge dimension (factual - meta-cognitive) are directly taken from the revised Bloom's taxonomy.

While using the taxonomy can be difficult and even the categorization of education objectives and methods requires experience its use in programming education can be beneficial and has been increasing (Sorva 2012 p. 20).

2.4.2.1 Application of Revised Bloom's Taxonomy Is the Search of Synergy

The taxonomy is an analysis tool for the structure of curriculum or a course and discusses many subjects, special focus is put on *alignment.*

> *Alignment refers to the degree of correspondence among the objectives, instruction, and assessment* (Anderson & Krathwohl 2001 p. 10)

While taxonomy leaves the choice of methods to the teachers by necessity, one possible way of avoiding the misalignment would be the usage of assignments that closely resemble the real life tasks in programming. Unfortunately this is much like throwing a swimming school pupil into deep water and seeing if they stay afloat. In this extremely complex environment the cognitive overload is extreme, as is the probability of failure. Even worse, the failure is highly uncomfortable to the student, which translates into a high cost in terms of student's motivation.

To be able to use these highly aligned assignments, without equally high cost in cognitive overload and risk of failure, scaffolds can be used to provide safety nets and prevent failure and the perception of “drowning”.

2.4.2.2 CS1 Studies Align Poorly with Fixing Programming Errors

What makes this taxonomy so interesting in terms of resolving bugs, can already be seen in the taxonomy table. A CS1 student is most likely working his way to remember and understand factual and conceptual information which are in the lower end of cognitive complexity and abstraction (such as trying to remember the basic syntax of the used programming language). However, tasks relating to detecting, locating and fixing bugs are in the opposite end of the taxonomy. Detecting or locating bugs requires skills of analysis. Locating bugs in any effective way, requires self-analysis, planning and strategy (skills located in the metacognitive category). Fixing a bug that is not simply a typing error (e.g. control flow bug), could easily require creativity. Evidently there is a large and clear dissociation between student capability and error recovery difficulty.

In simpler terms, performing a programming assignment requires much less skills, than fixing the student's ill fated first attempt. Those that are lucky to thread the right way at their first attempt can get through relatively easily, but those who happen to take a step in a wrong direction, can get trapped without any possibility of escape.

2.4.2.3 Teachers Encourage – Compilers Do Not

A significant difference in evaluating student's work is also evident. In the usual case, the student's learning is evaluated by the teacher. Regardless whether this means evaluation of verbal answers during class or written answers from an examination, the evaluation process includes a highly trained professional, or at least an assistant, both of which can adapt and circumvent many of the failings in the student's answer, and even correct the student's error with immediate feedback. In many programming assignments this is contrasted with the compiler's (or any other automated evaluation system's) inherent inflexibility. Where the human will seek correctness, the machine will seek error. A very bleak and dangerous environment indeed.

2.4.3 Helping Students Past Unique Problems

Constructivism provides excellent theoretical backing to describe the learning phenomena, but it also works on a very high level of abstraction and provides little in form of tools to facilitating the learning itself. One of the views associated in constructivism is the requirement for knowledge to be *viable* (i.e. useful) in the context and task at hand. Even more so, it emphasizes the uniqueness of each solution for each combination of context, learner and piece of knowledge being learned. The requirement of usefulness in a largely unknown context creates a large variation that makes our efforts for focused problem localization and solving inherently problematic.

Lectures can be used to cover subjects which are poorly known by most students. However the exact “holes” in understanding are unique and student specific. Clearly circumstances must be provided, in which the students can (and will) individually learn the missing elements. Moreover, as noted previously, imperfections in previous knowledge cause a strengthening *confirmation bias* and can damage further learning.

2.4.3.1 A Mental Model Is Formed Regardless of Access to Correct Information

One of the basic principles of constructivism is that the learner will inevitably construct a mental model of encountered phenomena and the construction of this model is done for the purpose of *viability* i.e. the model provides utility for some currently desired purpose.

Ben-Ari argued (Ben-Ari 2001 p. 56) that the two following arguments do not appear in natural sciences and must be taken into account in CS1:

1. *A (beginning) computer science student has no effective model of a computer.*
2. *The computer forms an accessible ontological reality*

A computer can not be understood, if no accurate model of its operation is known. While it is clearly possible to learn a correct mode, if no accurate model is presented the student will never the less require and create one. As the characteristics of this model are the basis of further knowledge, all inaccuracies of the model will inevitably lead to confusion and problems. Even more problems are created as the faulty model forces all further knowledge to be adapted to its faults, effectively twisting all later learned information into a partially erroneous form.

2.4.3.2 Correct Mental Model Should Be Provided as Early as Possible

Sorva (Sorva 2012) interprets Ben-Ari and makes a solid conclusion that to prevent this unfortunate process from proceeding the underlying model should be presented early in the education. Clearly this pattern of providing a foundation, before building is constructed is superior to trying to build the building first and trying to correct the broken foundation afterward. Moreover the computer is a data processing device and its model is a model of a process. This suggests using methods that suit to the specific purpose of teaching processes. Doing is learning.

2.4.3.3 Learning Should Be Based on Understanding

Carpenter and Lehrer (Carpenter & Lehrer 1999) state that learning should be based on understanding: "*The overriding goal of the classrooms should be the development of understanding.*" (p. 31), "*A mounting body of evidence supports the importance of learning with understanding from the beginning.*" (p. 32). This means that the students should not be required to master any level of skill, before starting to understand. Students who do not understand the subject perceive topics as separate entities and can only solve problems that have been explicitly presented and solved for them. Moreover Carpenter and Lehrer claim that once a subject has been learned through rote without understanding, it will hinder the student from developing this understanding later on. The reverse also holds true. Students who construct the knowledge through reflection and articulation, e.g. through their own activities, will develop a personal investment in building knowledge.

While Carpenter and Lehrer are discussing the teaching of mathematics and not programming, the two subjects share much in common, suggesting their work to be directly applicable to teaching programming. They focus heavily in understanding, which is described as playing a critical role in solving any complex problem and is a generative process of 5 parts:

1. Constructing relationships
2. Extending and applying mathematical knowledge
3. Reflecting about experiences
4. Articulating what one knows
5. Making mathematical knowledge one's own

There is a clear relevance to *cue-dependent forgetting* and *excessive cognitive load*. The problem is the complexity. In programming and mathematics, you need to be aware of a vast number of things that are happening "under the hood". As an example, a division requires the divisor to be verified as non-zero. If one would physically write the zero on paper, the strength of the cue would make forgetting the rule very difficult, yet when variables hide the actual values, this very basic rule is still easily forgotten. In programming, *program dynamics* takes this one step forward with it's mutable variables that rapidly change in value and in complex patters. In essence, knowledge that is known becomes unavailable through added layer of complexity. Understanding the interactions between the concepts allows the more efficient management of working memory. Strategies can be used to retrieve relevant cues robustly from memory instead of allowing them to clutter the working memory and *chunks* allow vast amounts of data to be retrieved with each request.

2.4.3.4 Threshold Concepts – Should Be Prerequisites for Coding?

The persistence of old behavior and *savings* (a behaviorism term for the faster learning speed associated in learning new knowledge that is connected to older knowledge, even if the older knowledge might even be already unlearned) can lead us to speculate on larger or more complex knowledge structures underneath the surface. One approach to this, is the exploration of *threshold concepts*.

The definition of threshold concepts is still developing, but they are likely to have the following 5 qualities (Meyer & Land 2006 pp. 7-9):

a) *Transformative* – Understanding changes the way the subject is perceived
b) *Irreversible* – Forgetting or unlearning the concept is unlikely and difficult
c) *Integrative* – Exposes previously hidden interrelations
d) Bounded – Forms boundaries between conceptual domains such as academic disciplines
e) Troublesome – Difficult to learn (appears counter intuitive, alien or incoherent)

It is easy to accept that if an abstract piece of knowledge is highly connected to many other subjects, its correct understanding would not only speed up learning the surrounding knowledge, but that its incorrect understanding would also delay or corrupt any such learning process. It would certainly help to explain the extreme learning variation between students in CS1 courses (Sorva 2012 p. 111).

If we accept that a skill should be understood before it can be used, clearly threshold concepts should be on the list of "required to understand", before any (but the most simplest) programs should be written.

2.4.4 Errors Are Inevitable

Errors can be described as: "*some knowledge structures are overly general (and hence become active inappropriately) and that decision making is a probabilistic choice among all active structures. These assumptions imply that displacement errors will necessarily occur.*" (Ohlsson 1996) (Displacement error in this context is the erroneous usage of the overly general knowledge.)

In the case of a novice, the usage of this overly general information is necessitated by the obvious lack of specific knowledge, on which to base more valid reasoning. Even if the required knowledge has been acquired, its usage might be lacking, i.e. previous (faulty) knowledge might sometimes be preferred because it has not been fully displaced by the new knowledge. As a result the debugging skills of a beginning programming student are very poor (Fitzgerald & Lewandowski 2008).

2.4.4.1 Bugs as Solutions Instead of Problems

CS1 students inevitably have poor and outright incorrect information on which they base their actions. To mend the flaws in their mental models, the flaws must first be located. One way for this is to put the students into positions in which their flawed information leads to a clearly erroneous result (i.e. a cognitive conflict). This cognitive conflict can result in a process of reflection and critical thinking (Ginat 2013) (Borasi 1996).

While the bugs are a serious problem in learning to program, it is beneficial to investigate the reverse statement. The severity of the problems tells us that a fundamental skill or a set of skills might have been missed in the teaching and / or learning process. While the bugs can be used to guide us to the problem at hand, they might take us even further in providing custom teaching opportunities that focus to the individual needs of that particular student. In essence, can we turn the table and transform programming errors from a difficulty, into a tool for more effective and easier teaching?

Ginat provides a demonstration of using error examples in teaching (Ginat 2013). Their approach was **not** to focus on problem solving (i.e. assignment completion or *operational view*), but on understanding programming language features and to a lesser extent computer model comprehension (*declarative view*). This was performed by presenting short example Java program code, which had been written with erroneous understanding of some vital programming concept.

One such example used object reference (i.e. address) comparison with "==" operator to compare if two objects' data are equal. Here are the most relevant lines of the example.

```
Point p1=new Point(1,2);
Point p2=new Point(1,2);
if (p1==p2)
  System.out.println("equal");
else
  System.out.println("not equal");
```

Their approach was to present the example program to the students, ask what the output would be, discuss both answers and explain both answers. Fittingly the number of discussed errors was lowered and the understanding of the related programming language concepts increased, compared to the control group, but the course attitude questionnaire also showed highly positive reaction from the students.

> *In conclusion, we offer some suggestions for tutors, based on our experience: 1. Identify erroneous conceptions of your students; 2. Address these conceptions with a constructivist approach, while trying to reduce knowledge gaps; 3. Develop activities such as those illustrated here; 4. Embed the activities in selected places in your CS1 teaching; 5. The activities may involve self-explanation, working in pairs, and class discussion; 6. The class discussions will involve a process of error attribution and reflection on both erroneous and correct solutions; 7. The discussion will conclude with accurate declarative specifications of the studied terms and notions; and 8. Throughout the activities and the discussion, try to raise the students' enthusiasm so they will feel they are both learning and having fun.* (Ginat 2013)

Clearly there is difference in presentation between using an example, which is the focus of discussion and for the student to come across the problem while trying to complete a programming assignment. The context is much more favorable to the investigation of the error, when it is the task itself and not an interference to a task you are trying to perform.

However, evaluation of assignment success or failure offers practical value. The correctness of a structured answer can be evaluated more easily and can even be automated. It is much more time consuming to evaluate if a student's perception is based on understanding the concept deeply (*declarative* view) or just performing the required steps (*operative* view) without particularly understanding their meaning.

Moreover, going through problematic subjects is time consuming. If the problems are selected for the whole class at a time, students who have no problems with the selected problems, will possibly benefit very little.

2.4.5 Dual-Coding – Improves Knowledge Encoding

Dual-coding theory as first proposed by Allan Paivio (Paivio 1971) and has been described as "*one of the most influential theories of cognition this [20th] century*" (Marks 1997 p. 433) (Thomas 2013). In short, dual-coding theory is an effort to explain, how addition of visual memories can significantly increase the learning and retaining of verbal memories, a phenomenon known already at the time of *Cicero* as the *Method of Loci.*

Dual-coding theory focuses on identifying two distinct, but highly interconnected main classes of mental representation[3], verbal representations and visual representations. These systems are independent and complete in the sense that they can perform all basic functions of memory (form, retain and retrieve memory traces) by themselves. However, the systems can also form associations into to the representations within the other systems and activate those external representations. The memories that can be evoked through both systems have much higher chance of being retained and retrieved, than representations that rely on only one of these systems. (Thomas 2013)

The theory has been contested by an array of theories that are commonly called as Common-coding theory, some of which also seem to be conversing with computational models of brain and thought. Common-coding theories bring more types of systems of mental representation or combine them all to just one, which undoubtedly has its merits, but also dilute the dual-

[3] While dual-coding calls these representations as "codes", this work refrains from using the term, as the word's already high and differing usage in Computing might result in unnecessary ambiguity. For the purposes of dual-coding theory, the word representation is used instead.

coding theory's core achievement of identifying visual representations in addition to verbal as the two capital types of information exchange and processing.

2.4.5.1 Visualization – How to Effectively Implement Dual-Coding

While dual-coding provides clear benefits in knowledge encoding, these benefits have been difficult to procure in CS1 context. Work done in *algorithm visualization* research provides us with tools on how to implement dual-coding in an effective way.

A meta-study of algorithm visualization effectiveness concluded that studies into algorithm visualization effectiveness are mixed, but the mixed results are due to differing success in application, instead of mixed effectiveness of visualization techniques in general (Hundhausen et al. 2002). If employed in a proper way, the algorithm visualization is therefore a highly effective catalyst for learning. "*Our most significant finding is that how students use AV technology has a greater impact on effectiveness than what AV technology shows them."* They also noted that the visual quality of the visualizations mattered little and that the results of plain viewing compared to those of conventional learning materials. Significant benefits were gained only through engaging students in interactive usage of the visualizations such as: *what-if analyzes of the algorithm behavior, prediction exercises and programming exercises. "Notice that, in such cases, rather than being an instrument for the transfer of knowledge, AV technology serves as catalyst for learning."*

2.4.5.2 Obstacles in Adopting – A Tool Must Be Easy to Approach

In addition to being efficient, a tool must naturally be adopted and employed before any benefits can be gained. Why educators do not employ AV technology (Hundhausen et al. 2002):

- *They feel they do not have time to learn about it.*
- *They feel that using it would take away time needed for other class activities.*
- *They feel that creating visualizations for classroom use requires too much time and effort.*
- *They feel that it is simply not educationally effective*

When we combine these with problems of the visualization effectiveness, it means that teachers must find tools approachable and useful to use them, and students must find tools approachable and useful to interact with them. While separating these two requirements between the groups might seem redundant, the two groups (teachers and students) are clearly very different in both goals and methods of operation. As a result, all the four requirements must be satisfied before significant results can be expected.

To provide a more concrete approach, these three steps need to be considered for all user groups:

1. **The features must be desirable by the user.** This is clearly the hardest requirement as the desirability of the features depends on many dynamic factors, is difficult to predict, and should optimally be based on continuous user testing and feedback.

2. **The features and their benefits must be presented in an easy and attractive way.** This creates the goals and motivation required for expending resources in trying out the tool. Possible ways include attractive presentations showing usage, benefits and features. Instructions for Installation and usage can also be made simple and attractive. Naturally all provided goals must be reachable i.e. the promises made to provide the initial motivation must be kept.
3. **The resources needed for reaching the goals must be minimized.** This can be accomplished by clear instructions and minimizing the required steps through design and user case testing. Unexpected events in installation can require significant effort, can be displeasing and therefore should not exist. The user should know all required steps, and these steps should be easy and fast. Clearly detailed up-to-date step-by-step instructions are needed. Installation should be automated and should not contain questions. Options can be provided, but only when explicitly desired by the user. Prerequisites should be as few as possible. Once the installation is complete, the same pattern should continue. During the first usage experience, the user should be guided and should not encounter unexpected events. Experimentation should be encouraged by providing an easy way of undoing the unwanted steps (*a domain of gains*). To avoid surprises, help should be provided within the software and not require potentially unavailable external materials.

2.5 Designing Intelligent Tutoring Systems

The design of ITS (intelligent tutoring systems) has been a long ongoing effort and has much to offer in terms of design principles. Anderson et al. provide one such list of eight principles (Anderson et al. 1997 p. 871):

1. *Represent student competence as a production set.*
   The decomposition of these production rules allows the student to understand what is needed or must be taken into account before the step can be taken.
2. *Communicate the goal structure underlying the problem solving.*
3. *Provide instruction in the problem solving context.*
4. *Promote an abstract understanding of the problem-solving knowledge.*
5. *Minimize working memory load.*
6. *Provide immediate feedback on errors.*
7. *Adjust the grain size of instruction with learning.*
   Small steps for beginners, larger for more experienced.
8. *Facilitate successive approximations to the target skill.*
   The student's problem solving skills gradually grow up to real-world problem solving requirements and the **need and availability** for scaffolds should gradually go down.

They also focus on the pedagogical importance of the type of feedback given to the student and its timing:

1. *Feedback should be provided at a time when the relevant information (problem state and action consequences) can be communicated effectively to the student.*
2. *Feedback should not change the requirements of the task.*

3. *Feedback should not disrupt performance of the task.*
4. *To optimize learning of the target skill, as measured in elapsed time rather than in elapsed problem solving exercises, feedback should be presented as early as possible.*

As expected, the design principles and pedagogical advice presented here fit well with our own conclusions on design principles drawn directly from cognitive psychology. They also present guidelines on how errors should be informed to the users and how the student should be *reteached* on the topic instead of just the particular misconception: *The correct analysis should be administered in at least three or four stages:*

1. *a reminder of the problem solving goal*
2. *a description of relevant features of the current problem state and the desired goal state*
3. *a description of the rule for moving from the current state to the desired state*
4. *a description of a concrete action to take*

3 Conclusions from the Literature – DAPS Requirements

This chapter first formulates *the persisting dissonance behind CS1 drop-out rates*, the leading finding in this work. The rest of the chapter then proceeds into translating the found theories into design requirements for Eclipse DAPS.

As mentioned in chapter 1.3 DAPS is a proof of concept ITS that aims to reduce *persisting cognitive dissonance* among CS1 students. It does this by first establishing productive interaction with the user through an easy and familiar text based dialog that imitates conversation. The productivity of the interaction (*the added value*) to the student is provided by the debugger and visualization features. During this interaction DAPS helps the student to maintain motivation and to avoid high dissonance traps that could derail the learning process or even push the student to abandon the course.

3.1 Persisting Cognitive Dissonance Behind CS1 Drop-out Rates

Conflicts such as bugs are an inevitable part of the natural learning process. Cognitive dissonance is the invaluable drive to find solutions for resolving those conflicts and it does this with a strong push type motivational force. Especially in less working memory intensive tasks that have relatively easy solutions, dissonance can be a purely beneficial motivator. However, the disruptive *dissonant* or irrelevant thoughts (e.g. noise, closing time limit, disagreeable lunch) created by dissonance claim some of the working memory capacity (chapter 2.3.1.4) and can prevent recollection of required task relevant memories. This results in a reduced performance in complex working memory intensive tasks. With every additional difficulty encountered by the student, the performance is further degraded.

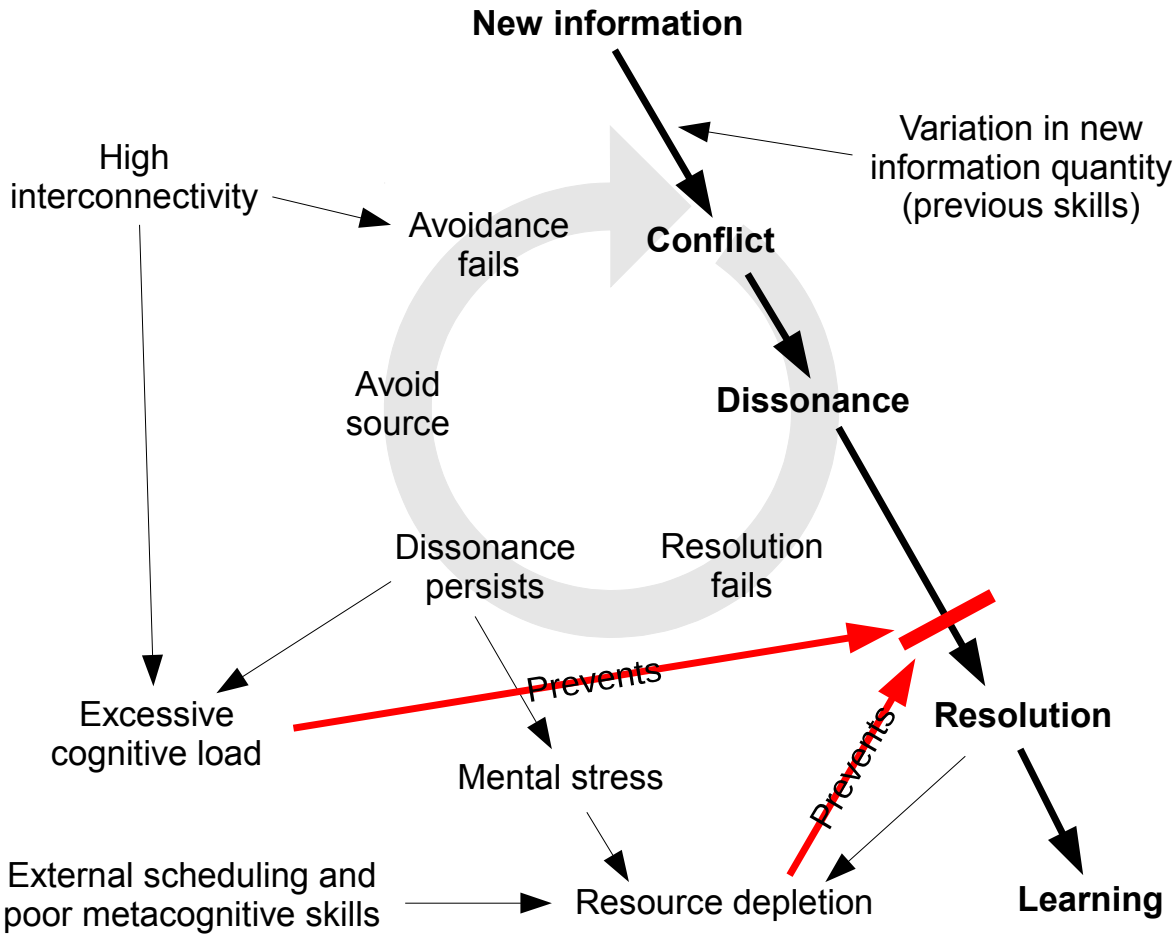

Illustration 3: A vicious circle formed by accumulating dissonance consisting of three harmful feedbacks: Resource depletion, Excessive cognitive load and Avoidance failure. Notice that two of these feedbacks are caused by interaction between high interconnectivity and persisting dissonance.

In addition to dissonance the high interconnectivity of CS1 prevents dissonance management through avoidance (chapter 2.2.3.8) and can lead into a vicious cycle of constantly increasing persisting cognitive dissonance. Tragically the reasons behind the escalation are most likely temporary in nature. In other words, the problem is a local aggregation of stochastic disruptions that by chance manage to reach critical strength and to become a self sustaining

vicious cycle of harmful feedbacks, essentially forming an ever strengthening trap for the unfortunate CS1 students. If this conclusion is accurate, a large portion of those students could be rescued back to the normal curriculum flow with relatively few but carefully considered interventions.

3.1.1 Helping to Dissolve Cognitive Dissonance

Cognitive dissonance is created by conflicting information. As many parts of the student's knowledge are flawed by necessity, allowing the student to rectify dissonant errors as easily as possible will reduce both erroneous knowledge and the resulting dissonance. Even if dissonance is used to motivate the student in the form of presented challenges, it should never be allowed to persist. As we have concluded before, persisting cognitive dissonance is highly destructive.

For example providing context dependent help topics allows a student to focus information searches to relevant subjects regardless of poor understanding of the overall picture and avoids both the wasteful brute force search and the associated motivational cost. The reduced dissonance allows better concentration on the task at hand and reduced mental stress level, both of which enhance the encoding process in acquiring and storing (maintaining) new cognitions.

Moreover, when information is provided based on the student's own learning process, the student can focus on dissolving the most dissonant problems first, without having to endure their burden until externally defined schedule provides the information needed for properly dissolving the dissonant conflict. This does not necessarily mean that the student would benefit from choosing the order in which subjects are taught, as teachers clearly have superior overall picture, but it does mean that within this externally designed superstructure the student still needs to maintain fairly dissonance free understanding of the subject. As a simplification, one could say that **the states with too high dissonance automatically lead to escaping from the state and avoiding the circumstances that imposed such a destructive state on the person**. One way of conceptualizing this would be running the two processes side by side: the first process which teaches selected subjects in superior learning order (curriculum and course teaching), and a second process that maintains student's consonant mental state throughout the process by clearing out dissonant conflicts and preventing them from accumulating (assistance in encountered problems). This second process is where assistance tools can make their impact in guiding the student past pitfalls, while in the same time allowing the student to choose the direction and the problems to be fixed.

This preventive approach gives additional benefits in reducing the need for unlearning incorrect information. Dissonance is extremely uncomfortable and as such its resolution is non-voluntary but is forced by the person's own cognitive processes. If no proper way is available, poorer methods are used by necessity. Timely resolution will therefore liberate the student from having to fabricate temporary “quick fixes” that could further derail the learning process.

3.1.2 Deadlines in Inducing Beneficial Dissonance

Deadlines are excellent sources of dissonance, which also explains their efficiency in motivating action. In case of a complex or a large task, deadline poses the compelling push motivation, but the student must independently develop the metacognitive management skills

that are needed for the organization of resources and the performing of the required tasks to succeed. As these skills are vital, students must also increasingly demonstrate them as their studies progress. However, in the case of introductory courses such as CS1, the focus should clearly be on instruction instead of requiring demonstration from the student. An example of easier management would be a requirement of spending a fixed amount of time instead of requiring a fixed end result.

Explicit induction of dissonance (such as a deadline) is least harmful with easy tasks that can be guaranteed to succeed (such as participation to an exam or lecture). For example, the participation to an exam is most likely easy, even when passing it might be hard. Examples of metacognitive instruction would be requiring plans and self designed schedules to be submitted before actual work can start, and gaining evaluation and feedback on the quality of the submitted plans. Another example would be to identify dissonance as the factor behind poor performance and subsequently employ active cognitive techniques to reduce dissonance.

3.1.3 Persisting Dissonance – Considered Harmful in CS1

Cognitive dissonance creates a powerful *push* motivator for solving problems. However, as this motivator is negative in nature, a conflict that remains unresolved can lead to problems far greater than the simple failure of the affected task. *Cognitive dissonance* leads directly to higher mental stress and discomfort, which continues until the conflict is solved in a satisfactory way. If sufficient resources are not available, the conflict will continue taxing these mental resources and causing problems for the student, resulting in a degraded learning results.

What makes cognitive dissonance especially difficult is its unpredictable nature:

- High variation between persons: It is directly dependent from previous *cognitions*, social contacts and supporting networks. For example whose opinion is valued and consulted in case of problems.
- Problems are actively concealed: In order to reduce dissonance, the person is actively belittling the true source of the problem and is reluctant to acknowledge the existence of the problem.
- Persisting dissonance will lead to strengthening avoidance behavior, reducing opportunities for finding the solution.
- Sufficiently strong dissonance forces the person to retreat from seeking “correct” views, into seeking views that reduce dissonance (strengthen own view regardless of own view's correctness), and devalue persons who promote the “correct” view.

All of these factors increase the risk of problems rapidly escalating into abandoning studies. The risk of rapid escalation would also imply that only preventive or early interventions are possible (or at least cost effective), further limiting available methods and their effective time window.

3.1.4 Variation in Skill – Adaptation to Change Requires Time

Motivation can increase expenditure of resources to the current task, but regardless of its nature (positive *pull* or negative *push*) the resources are still expended. Hence the person's limited resources dictate the upper limit to the benefits of motivation.

The cause of *cognitive dissonance* is the introduction of conflicting new information and regardless how well these conflicts can be cleared out, those who have already worked on the subjects clearly have less work ahead of them. Differences in initial skill level (as well as differences in other resources), lead to differences in dissonance. Those that are behind, are pushed harder by the dissonance, and while the dissonance provides them with strong motivation, the work still needs to be done. As sources of dissonance variate between students, better resource management can help in preventing local maximums from developing, but in the end, if the same time and other constraints are presented to all subjects and same results are required, there will always be a fraction for whom the task is simply impossible.

This leads to two conclusions:

Firstly, if all (or nearly all) students are supposed to manage through CS1 and preferably even have a positive reaction to the subject in the end, the teaching, the skill requirements or the timetables can not all be equal between the students. The initial variation between the students necessitates stronger support, longer time or lesser skill requirements for initially less skilled students.

Secondly, in some cases (such as entrance exams), this variation is intentionally left without compensation, as the goal is to weed out those who have less chances of managing through. However, was this difficulty intentional or not, being prepared for the chance of failure is important, and this preparation should be made possible (for example by providing statistics of drop-out rates). Students should be informed of difficult exams or exercises, so they can prepare for the high failure rate and understand that the task is harsh and demands previous skills or high devotion. Otherwise there is a clear risk of students entering without any real chance of passing, but with full expectations of doing so, leading to a clearly damaging outcome for the student and into a waste of course's resources. This is especially important for young students, as their undeveloped metacognitive skills are challenged by the new environment and can not yet provide protection against unfavorable outcomes. Preferably, easier (higher rate of success) options should be provided and promoted to those that are doubting their chances beforehand and to those who are too late in realizing the difficulties and want to switch into a slower schedule.

One might also argue that as CS1 is an introductory course, it naturally has a very high variance in initial student skill and instead of trying to provide efficient teaching in the subject, it should concentrate into reducing the high variance by filling the most severe gaps in student skill. The more efficient teaching could then proceed on later courses when the initial skill variation has been reduced to manageable levels. While this would make the course “too easy” for most of the students, it would also not “crush” those that want to try CS1 without any previous skills. However, while course demand reductions would clearly reduce sources of dissonance, they are outside the scope of this work. Moreover the necessity of such decisions is also in question, if the same learning results could be obtained simply by better individual management of dissonance. Instead of posing additional restrictions to the course design, DAPS aims to address these problems with as little interference to the curriculum or course decisions as possible.

3.1.5 Metacognitive Skills – Knowing What to Expect

Better *metacognitive knowledge and skills* (planning, monitoring, evaluating), can help in alleviating some of the cost in cognitive dissonance through rationalizations and strategies such as “I don't need to understand this matter yet, I have time scheduled for learning it later.”, but these skills are also at their weakest among the CS1 students. These same skills are much more developed in the course personnel side, making the estimation of student's difficulties even harder, and possibly undervaluing the perceived problems that the students encounter. Self-regulated learning (metacognitive) skills help in understanding your own circumstances and positioning yourself accordingly, the assistant tools can remind the student of the reasonable self-evaluation. This allows the student to reduce damage from ordinary failures, by understanding their near non-existent effect to the whole course/degree.

3.1.6 Facilitate Acquisition of “Missed” Prerequisites

While the consideration of prerequisites seems to fall more on the context of curriculum design than DAPS, this need not to be so. As understanding is a generative process, it has no fixed state and develops constantly in a fluid fashion. What is “missed” depends heavily on purely stochastic factors, such as having a flue on a particular day. Moreover the meaning of “prerequisite” is relative in a network of interactive concepts. The incorrect prior knowledge could be corrected at the first moment it is used to limit its detrimental effects on learning.

Encouragement to understanding the relevant subjects can be provided in any situation in which the student has the required resources, most notably the time. Whenever a flaw or a hole is detected, DAPS can ease its investigation with easy referencing and encourage developing understanding over immediate solution to the problem. Each time the student returns to past materials, the same information is considered from a more mature perspective and additional understanding can be gained, even when the referred text remains the same.

Most importantly this approach alleviates the need for prior identifying of the problems. While prior identification and direct instruction would be more effective, it is not always possible. For example, the definition of *threshold concepts* (chapter 2.4.3.4) remains elusive. DAPS would in any case need to evolve to cover most basic subjects. As the threshold concepts are highly connected, they would undoubtedly be covered as well, but in the context of DAPS, explicit knowledge of their identity is not needed. Concepts can be added simply because the students perceive them as puzzling. The inherent complexity in threshold concepts suggests that without explicit provisions in the curriculum, they would take longer to understand. **Because of this, threshold concepts would be likely to enrich into the group of “missed” concepts.** In essence, by easing the acquisition of “missed” prerequisites, DAPS would directly improve the understanding of threshold concepts.

3.2 What Can DAPS Do? – A Numbered List of Requirements

The rest of this chapter contains requirements for DAPS. They are first presented with detailed rationales and finally listed as a chart (chapter 3.2.16). Each requirement is numbered and end with a tag that tells the current implementation status of the requirement:

- \- unimplemented
- \+ partially implemented
- ++ fully implemented

Many higher end features would require individual student's working to be monitored (and profiled). Currently most of the presented requirements such as monitoring, visualization and assignment packages are unimplemented. This list aims to propose as many viable future feature options as possible, and DAPS could be used productively with only a fraction of the presented requirements.

3.2.1 Motivate to Learn, Not to Pass Assignments and Tests

Success in assignment or test can be motivating, but presents an extrinsic and as such a dangerous and volatile form of motivation (chapter 2.2.1.3). It is directly tied to external evaluation to which the student has relatively weak control, and will backfire the very moment these good evaluations stop coming. As motivation is a constantly needed resource, it is needed also when difficulties are encountered. Because of this motivational sources should persist during adversity and in CS1 should preferably be guided towards learning and excelling in the programming itself. Encouragement to get good grades is already strong, as grades are used in important decisions concerning the student. DAPS should ignore or replace the extrinsic motivations and focus on strengthening intrinsic motivations.

1. Dialog should promote improving skills and discourage to worry about grades. Concentrate on improvements in skill, not the absolute skill level. Encourage to find problems and solve them. +
2. The focus should be on the proper procedure, not the end result. This can be done for example by task design by breaking tasks down into smaller tasks or by asking the student for a plan on how to tackle the problem, instead of the end result. -
3. Metacognitive instruction and knowledge can be included. An example of this would be information of intrinsic motivation's superiority over extrinsic. -
4. If student is frustrated due to assignment not progressing, DAPS can remind that the only reason for the assignment is to allow the student to learn. Passing or failing is meaningless for the end result and only provides procedural information. -

3.2.2 Task Performance ≠ Learning – Time and Effort = Learning

Failure in a task is unfortunate as it most likely drains the student's motivation, but much might still have been learned. Equally, success in a task does not mean any learning has happened. Effectively evaluation has two parts: 1) it can estimate student's skill or 2) it can estimate student's effort. The students inevitably ignore the first (due to their poor metacognitive skills) and considers evaluation only as the evaluation of themselves and their effort. This means that from the student's point of view, if the students consider themselves capable for the course, the evaluation is fair only if it depends from the resources invested on

it. Large effort should therefore bring good evaluation regardless of other concerns.

5. DAPS never evaluates student skill, but needs to track student's effort. -
6. Track the time the student uses with each task. (Data remains local.) Periods of inactivity need to be ignored without troubling the student. -
7. Time and effort correlate to learning and should be evaluated and rewarded accordingly. -
8. Rewarding can be as simple as complementing the student through dialog. Discrete social rewarding schemes could also be employed. +
9. After sufficient amount of time has elapsed, DAPS should remind the student not to overexert themselves. This can be considered as both encouragement and supporting student's time management skills. Regardless of progress, overexertion reduces learning and drains motivation. Student could also be stuck and wasting both time and motivation. -

3.2.3 Students Need Help in Evaluating Their Own Performance

Students should know what is expected of them and which expectations are reasonable and which are not. Programming involves a highly complex and broad set of abstract skills. A glance at *Bloom's taxonomy* and the discrepancy between realistic and fictional CS1 expectations should emerge. Students should understand that their variation in success during CS1 is much more credited to their past than their present efforts. They should understand that the ones “who have been coding since seven” have an edge that can not be cached up in just few months or even years. Those who are new to the concepts of programming, have much more to study than the simple syntax of their first language. Studying your first programming language has very little to do with that first language. This can clearly be seen from for examples how an experienced programmer can “learn” a new language in just one day. Something that would not be possible if the actual semantics of the language were the difficult aspects of the task.

10. Short before and after tests can be used to display the student how progress is being made. Ready made assignments or bigger tutoring paths can include this type of short quiz sections. -
11. Monitor students' overall time usage across the whole course. (Requires user identification, time tracking and central data collection). -
12. Allow the student and course personnel to access this tracking information. Motivate the student by showing how much progress has been done. Implementation could be a pop-up accomplishments window accessed through a trophy shaped button in the interface. -
13. Discourage speed competition. Speed comes from already knowing the subject. Learning comes from spending time. -

3.2.4 Deadlines for Large or Difficult Tasks Can Be Harmful

14. Allow student time usage to be used as course requirements. (This could allow removing some assignment deadlines.) -

15. Create a ready made CS1 package of assignments that can be employed by a course or an independent student, in which the schedule monitor tracks and informs the student of the time expended and progress made. -
16. **A failed deadline, as any other severe failure, is a warning sign.** The student should be asked, why the deadline was missed (and assisted to complete it). If progress monitoring is implemented, the tool should inform this to the course personnel immediately. -

3.2.5 Repetition Is Vital

From *Behaviorism's classical conditioning* (chapter 2.2.2.1) we learned that while *extinction* of learned behavior does happen and "bad habits" can fade, they are extremely persistent. Any perceived benefit from the extinct old behavior can immediately bring it back to its full strength. The old behavior is much stronger, and if conflict between the two should emerge, in the absence of strong support to the new behavior, the old (as flawed as it might be) will prevail. Still *behaviorism's* strongly backed up finding is that any behavior will become stronger through repetition. Enough repetition will end the "tyranny" of the old behavior and set the student free from the persistent error, but only with sufficiently large number of repetition. Repetition is the key.

17. Do not automate or automatically assist any steps the student takes or needs to take. Automation can disrupt flow and needlessly complicate matters, in addition of robbing the student from the learning experience. As the steps needed are in our case quite small, no practical need for automation exists. However next steps can be suggested. +

3.2.5.1 Repetition Must Be Elaborative, Distinctive and Cue Invariant

The quality of repetition is also important as shown by the *Levels of Processing* theory (chapter 2.3.2). Repetition must not follow the exactly same path every time, but variation in its surroundings, or in the way it is processed, must exist or benefits of the repetition remain negligible. Short, clear and distinctive processing of the subject in question provides time efficient and lasting memorization. The used *cues* (for example the terminology) must be reused or chances of recall drop dramatically (chapter 2.3.3).

While variation is important to understanding, same reference materials can still be used repeatedly. Here the variation comes from the changes in context and the student's development. Each time the same materials open up in a different way and are considered from a different perspective and can therefore provide additional understanding.

18. Same cues must be repeated across all DAPS material. This uniformity would be extremely beneficial also across the whole course (or even multiple courses) and DAPS should encourage uniform nomenclature by providing easy access to this dictionary (or use an external shared dictionary). This would for example allow a lecturer to just cut and paste relevant cue-elaboration pairs and put them on a side display for the duration of the lecture. This could ease lecture preparation, provide common nomenclature, perception of security to the student (as the reference stays there) and improve comprehension and memorization. +

19. The used dictionary (cue-elaboration pairs) must be easily and constantly accessible. After the cue and elaboration, a separate additional information section can be provided that can be extended to contain as much information as wanted. However, the compactness does not matter, as later on the student becomes more familiar in using external (more extensive) sources. -
20. The elaboration must be clear (short), single (no lists) and distinct (should not remind each other). -
21. The cue itself should be as distinct as possible. This inhibits cue-dependent forgetting and prevents cues from being mixed with each other. Indistinct cues can be detected from high number of cues, from cues being mixed with each other or from the poor retention (forgetting) of the cues. Quizzes or tests can be used to assess cue quality in addition to providing repetition and tracking information. -

3.2.5.2 Repetition Must Be Successful

Repetitive success is also vital in maintaining motivation, which allows the learner to push through obstacles even when failures and unwanted results sporadically occur. Confident perception of the task's success will increase motivation and allow more resources to be used in completing the task. Thus, a steady supply of successful subgoals can maintain higher working intensity (more resources committed) and longer time spent (before giving up).

For the students that have little or no prior positive experiences in programming (and are hence less committed), **CS1 is in critical position**, as it is among the first times the student encounters programming. If a high motivation can be built during CS1, the resulting commitment, can become a strong motivational resource in the following studies.

22. All presented goals must be reached. Tasks should be easy and any obstacle should be clearly achievable at most with reasonable extra effort. -
23. Each student's success in tasks should be monitored and intervention made if failures start to accrue. -
24. Individual tasks should also be monitored. If a task has high failure rate, it must be fixed or removed. -

3.2.6 Easy Access to Relevant Information

Easy access to relevant information is vital for repetition, memorization and understanding (chapters 2.2.4.3 and 2.3.3). While this can be achieved with separate materials, the dissociation between them will increase cost of acquiring relevant knowledge and hence reduce the frequency in referring to such material. Also the student is able to rapidly experiment and learn the difference between possibilities, when lacking or incorrect information can be easily remedied. Ease of access also promotes verification of already understood subjects and hence prunes out hidden problems.

While the tool can not help in all problems, the most frequent problems can be addressed and solved more easily. This reduces interruptions to repetition cycle and contributes directly to completing more subgoals and increases motivation. This increased motivation can then be reserved to combating the problems in which the assistance system is not able to provide assistance.

25. All needed materials should be easily accessible (for example through hyperlinks). This also provides a fallback strategy to the student in the form of going through the list. -
26. Good external sources of information can be linked or suggested. -

3.2.7 Avoidance – Prevent Avoiding and Complete Failure

Just as a deadlock will prevent a thread from performing any useful action, similar deadlocks can occur in people. *Approach-avoid conflicts* (chapter 2.2.2.2) demonstrate how there can exist show-stopping problems regardless of steady progress being made. Allowing complete failure can effectively hide a growing problem in one specific area or task. The problem grows, as avoidance forms into its dominant solution (chapter 2.2.3.7). This leads to the inevitable conclusion that if avoidance can not succeed in solving a particular problem, the attempt should be prevented from the start. However, as avoidance is also a resource management strategy, avoidance prevention must be done in a way that still allows the students to successfully manage their resources.

While demanding perfect performance in a test would be unreasonable, programming tasks can be rigid and can have little ground between perfect performance and complete failure. To avoid complete failure, there should be no failed assignments, only assignments that require more assistance. The failure in an assignment should be a clear indication that help in that area is needed, and progress should not be demanded before this broken foundation is corrected. Successful results should naturally reward the student with congratulations on the success to uphold their motivation, but failure should require much more attention. While there are many more reasons to an occasional failed assignment than hitting an approach-avoid conflict avoidance behavior, these seemingly random failures can be the only warning about the formation of this danger. To prevent developing avoidance from hiding behind other random occurrences, all of these failures must be treated as potential signs of the danger. After the behavior is formed, the student will not only try to actively hide it, but will also actively resist any attempt at correcting the problem.

While a programming assistance tool can not replace human intervention, it can help in limiting the amount of these complete failure situations encountered by the students. As an additional benefit, reduced failure rates release resources into dealing with the more severe problems and could lower the bar in seeking help and assistance.

The tool can also provide some assistance in seeking this help or reduce the damage caused by the failure by maintaining positive feedback even when the assignment subgoal itself is doomed to fail.

To prevent avoidance from forming, the students should not be able to **avoid** or **completely fail** in any problems presented to them i.e. all problems must be faced and an option of graceful failure must be provided.

3.2.7.1 Prevent Avoiding – Prevent Avoidance Strengthening Through Repetition

This avoidance can be anything that removes the immediate need to complete the task. Examples could include: doing it later, doing something more important first or just simply failing. If at all possible, the problem should be solved as it is encountered and other matters

should be pushed forward.

However, while avoidance is a poor behavior choice (chapter 2.2.3.8), other realistic (as perceived by the person) choices must be present, or the removal of avoidance strategy, can lead to even poorer choice of behavior. In addition, care must be taken not to introduce rigid supervision or loss of freedom in the learning tasks, or motivation can suffer. While there is some contradiction between preventing avoidance and maintaining individual freedom, there are also non-contradictory methods available, such as providing assistance in completing tasks that otherwise would have been accepted as failed and hence not required to complete.

27. All tasks that are given to the student by DAPS, should also be completed. Ultimately personal assistance by course personnel can solve practically any problem. +

3.2.7.2 Prevent Complete Failure – Replace with Graceful to Reduce Damage

When a problem can not be completely failed, there is much less benefit in avoiding it. Graceful failure (lesser extent of the failure) also reduces the damage inflicted to the person.

Focusing to the process instead of the result may also provide protection from failure, by reducing its importance. It will shift the motivation from *extrinsic* to *intrinsic* to provide better motivation. **Accountability is also moved from the result to the process, which should reduce *selective exposure to information* (avoidance of conflicting information).**[4]

28. A long and hard effort should never result in failure. The damage caused by a failure is proportional to the effort (and motivation) expended and the failure after a hard motivated effort is highly damaging. This requirement is general in nature and methods used in other requirements (such as interrupting before too much effort can be expended or directing to ask for help) can be used limit the damage. +

3.2.8 Eliminate Deadlocks – By Always Providing a Fallback

If the invaluable repetition is prevented by problems (for example bugs), there can be no effective extinction of the old flawed behavior, nor effective absorption of the new superior techniques. The elimination of being stuck is a prime design requirement. Constant progress in the task itself is not necessary and being forced to search ways forward can provide a powerful motivator for learning. Still, when a problem prevents progress in the task, **some way of action should always be available** that can believably result in solving the problem and allowing further progress.

This also means that while success in the programming task can be a motivation providing goal, the assistant does not need to know how to solve the exact problem. It needs to provide a plausible way forward by providing the prime candidates for problems in the current conditions. While more abstract and therefore somewhat more difficult, this task is also less dependent on exact circumstances, and the exclusion of these exact circumstances allows us to limit the required interaction tree (dialog tree / network) into a manageable size. We can also get beneficial repetition without direct progress in the programming task itself, giving the assistant a much larger and easier environment to work in.

29. When a student gets stuck, there must always be a way forward: a list of links to reference material, guidance in finding the bug, etc. +

[4] The requirements are redundant with Motivational requirements, and are therefore skipped, but even when the requirements are redundant, this rational is not.

30. Remind the students, that if a problem bothers them and can not be solved even after several hours of trying, they should always contact course personnel for help before giving up and trying to forget it. -
31. “A panic button” in the interface to provide advice when all seems to fail. This should provide motivational support (encouragement), metacognitive knowledge (to defuse the mounting negative cognitions) and finally guide the student to seek course personnel assistance. -

3.2.9 Repetitive Failure Is Dangerous

Repetition is only beneficial, if the activity that is repeated, is beneficial. There is little benefit in repeating activity that associates harmful qualities to the work at hand. Even a successfully completed programming assignment might not be successful in teaching the associated concepts, or might even teach some incorrect or harmful ones, such as: “I got through, but I have no idea why this works. I am just glad to be rid of it.” Especially the sentiment: “I am just glad to be rid of it.”, is a clear example of the dangerous avoidance behavior.

If the problems lead to repetitive trying that too often leads to a failure instead of a success, the act of coding can be perceived as the behavior with negative result. As repetition provides a powerful learning method, this cycle of perceived unfavorable outcome from trying to code, can lead to the extinction of the attempt itself, and thus to the abandonment of the studies that contain such activity.

In teaching CS1 students to code, it should be paramount, that we do not teach them to quit.

The key to solving this problem of repetitive failure lies in its definition: The *perception* of the outcome. Conflicts will arise, and there will be bugs, but the outcome need not be unfavorable. We can remove some of the unfavorable outcomes by “softening the blow” and some can even be turned it into successes by guiding the student past the deadlock in a rewarding way.

3.2.9.1 Avoiding Dangerous Repetition

Avoiding unwanted behavior being strengthened, should be important in any learning activity. Providing assistance before encountered programming problem starts damaging motivation and causing frustration, is preventive in nature. While investigating these dangerous circumstances would be worthy for more detailed analysis, we can easily identify several symptoms of such situations, which are visible to the tool:

- Continuous recompiling with little to no changes to the code
- Continuous undoing of changes made
- Solving problems by always consulting someone else (no trust in own problem solving skills)
- Long pauses after unsuccessful test (for example compilation)
- Long pauses in general
- Long time in completing an assignment
- Failure in completing an assignment

Some of these examples describe situations where unsuccessful attempts are made one after another. Some describe situations, where problem has left the student with no idea how to proceed in its correction. The latter case is more difficult to analyze. It could be that the student is still searching other sources for information, but the progress of that activity remains unknown.

In short, any activity that is not producing beneficial repetition (i.e. constant successful completion of desirable subgoals) has the potential for providing harmful repetition. It should be noted however that the definition of a subgoal in this case is broader and not limited to curriculum subgoals. For example, telling one's friends about own success in a task and gaining positive feedback is highly motivating to the person, even when this activity clearly completes no assignment or other curriculum subgoal. However, as motivation is required for reaching curriculum subgoals, maintaining motivation it is actually a far superior subgoal to the person than those provided by the curriculum. Therefore the lack or failure in these diverse motivational subgoals is quite dangerous.

Once identified, these harmful circumstances should be interrupted by a clear but unintrusive offer of assistance and encouragement. This early prevention also reduces the risk of problems escalating into “deadlocks”. Care must be taken that, as the user is probably frustrated at the time, intervening has the danger of guiding the user's anger towards DAPS.

32. Identify harmful repetition and try to interrupt it in an unintrusive and considerate way. The dialog should for example contain a one-click option for “never bother me again”. -

3.2.9.2 The Perception of Failure and Success Can Be Influenced

Prospect theory (chapter 2.2.4) tells us that the perception of *domain of losses* reduces productivity by favoring improbable and unproductive solutions, where as perception of *domain of gains* allows these risky or poor solutions to be avoided. By simply approaching the same task from the perspective of gains, higher *value* can be achieved.

Framing effect tells us that we can create a perception of gains simply with a suitable wording. For example an assignment might be be graded with 100 - errors, or 50 points + bonus points. Instead of struggling to avoiding penalty, the perception of superior accomplishment can be created.

33. DAPS responses should be overly positive and encouraging. (But not silly.) +
34. DAPS responses should be honest. A clearly negative matter can not be stated positively, without being false. Instead of focusing on the negative matter, seek a valid and relevant positive matter. -
35. If negative outcome is clear, it must be handled. In this case, the user can be reminded of the insignificance of the failure and that regardless of the failure many other goals were achieved. -
36. DAPS responses should not contain negative connotations. +

3.2.10 Reducing Extrinsic Cognitive Load

Cognitive load is likely to increase harmful effects form dissonance and avoidance behavior. Moreover, CS1 has a high *intrinsic cognitive load* which by definition can not be reduced. This increases the need to reduce *the extrinsic cognitive load*.

37. Instead of referring to a code line number (indirectly), refer to it with color change or arrow (direct reference). -
38. Prevent *split-attention* by keeping all DAPS related information in dialog window. Referenced material can be displayed in other windows and kept connected with a single line of text in dialog window that contains hyperlinks or with arrows that are shown when the pointer comes over the referring text. +
39. DAPS handles assisting through a large number of easy small steps. This allows concentration to one problem at a time. +

3.2.11 High Cognitive Load Necessitates Metacognitive Skills

The resource restrictions for the student come from many sources. One of the most severe ones, is the working memory. This limitation restricts the number of memories that can be simultaneously processed and by necessity subjects all remembering to push a previous memory out of consideration. So while new information is known, its application is subject to competition between other memories relating to that particular learning context. If the resulting high cognitive load (unavailability of working memory space) prevents concurrent recollection of some necessary piece of information, all required information can not be processed and correct conclusion will be hard or impossible to reach. In this case, *schema construction* is necessary, before problem solving can continue. While *schemas* can be used to lift this restriction, they are domain specific and hence *schema construction* requires considerable processing and understanding of the domain to provide this agility.

40. DAPS eases cognitive load by facilitating schema construction. Tell this to the student as metacognitive knowledge. Essentially, promote the importance of knowing the prerequisite information before attempting to go forward. Ask questions and allow the student to choose the direction, but offer references and tasks relating to prerequisites easily and often. -
41. To divide resource costs uniformly, DAPS can encourage user to work every day and for reasonable hours at a time. This could be implemented for example as a tip. More involved implementation could track the actual used time. Little effort every day is better than focused effort once a week. -

3.2.12 Erroneous Perceptions Must Be Readily Corrected

Constructivism tells us that all new experiences are built over the old ones. The new experiences are understood in the context of the old and the old experiences define the “vocabulary” that is used in understanding the new experiences. The tyranny of the old experience, can radically alter the new experience making it fit into the confines defined by the old experiences, thus twisting it into an erroneous form. Correct model should be provided and reached as early as possible (chapter 2.4.3.2).

42. DAPS must provide easy way of changing previous incorrect perceptions. (This is a general requirement that should be considered with all features.) Make finding the errors and misconception a positive goal. -
43. Old material must be easily accessible (previous exercises or explanations). -
44. Explanations must be easily accessible (hyperlinks). -
45. Previous steps must be easily accessible (undo/redo). -

3.2.13 Key / Threshold Concept Understanding Needs Verification

The difficult aspects are beneath the surface, such as *program dynamics*, which is a candidate for *a threshold concept* (Sorva 2012) (chapter 2.4.3.4). Understanding these underlying highly abstract and difficult concepts, transforms the thinking of the student in a profound way. This transformation is by definition a non-trivial endeavor, which takes much time and effort. As some students have already breached some of these difficult thresholds before they set their foot on the CS1, these students clearly have much easier time in performing tasks in subjects they already master. Direct comparison between these groups is clearly unfair, and represents more the amount of studying spent on the subject prior to CS1 than a good estimate for performance in the field of computing. The students, however perceive only the end result (the circumstances on the CS1) and as they are unaware of the difficulties beneath, could easily credit the success or failure to their own CS1 performance, and as a result, incorrectly judge themselves unfit for the task. This incorrect self-evaluation is an example of failure in students' metacognitive skills, which can lead to the perception of failure even when fast progress is being made.

46. When student has severe difficulties, tell that the cause might be in not understanding key concepts. Ask questions to test this. If the student can not answer the questions even with DAPS's help, instruct the student to ask course personnel's assistance. This contacting should be as easy as possible. -

3.2.14 Visual Representations Can Improve Learning Efficiency

Visualizations and dual-coding (chapter 2.4.5) provide us an easy way of improving teaching and learning efficiency. By employing two complementing memory representations we can increase the chance of recollection and the correctness of the recollected information and thus increasing the chance the student will employ the new (and still “weak”) information. The increased recollection will also improve the quality of the recollected information and increase the chance of successful application, which in turn can start the repetition and the positive reinforcement of the new and improved behavior. Visual components (of known syntax) can be faster to examine, than textual representations, and can ease interaction through easier testing of hypothesis'.

47. Create or use external debugger to gain access to dynamic runtime information, which can be provided to the user as easily accessible visualizations. +
48. The added visual information should be very fast to examine. No complex encoding. -
49. The syntax of additional visual components must be intuitive, but also be clearly textually explained (for example by tooltips) and preferably their efficient usage should be instructed by an accessible tutorial. -
50. Add numbering or arrows into the code display to visualize program flow, for example into an additional column. Loops can be facilitated by adding columns and circular arrows. -
51. Add variable data into the code display, for example into an additional column. -

3.2.15 Third Parties – Easy Access and Adoption, Hard to Let Go

Algorithm visualization research tells us how very important the ease of use is (chapter 2.4.5.2). The provided tools must be easily adoptable and have a minimal learning curve.

These tools have no funds for extensive marketing campaigns, so the tools will need to provide maximal positive and minimal negative experiences, to ensure their acceptance and application.

The time spent with the *visualization* correlates to the benefits gained. Similarly intention to learn has only very minor benefits to the learning process, the exposure itself is vastly more important. Therefore the tools should be designed in a way that increases the usage frequency and duration. Help should be provided fast, easily and unintrusively to increase the frequency of usage. Information about the possible solution should be available immediately, but a direct solution (i.e. correcting the problem for the user) should never be granted without at least a minimal investment in understanding the problem itself. These opportunities for learning must not be given up too easily.

Usability engineering promotes the importance of consistence. Our knowledge of cues gives strong support to this approach, as changing cues reduces recall, even when the newer cues were superior in strength. This also holds within learning materials provided, and while synonyms should be listed, only one set of terminology should be dominant. The CS1 student should not be faced with five different terms for each new concept even when a more skilled professional would benefit from the additional nuances provided by the larger vocabulary.

52. Interface design must be consistent (inhibits cue-dependent forgetting). +
53. User feedback should be encouraged. -
54. To promote adopting, features and their benefits should be presented in an easy an attractive way. For example You-Tube usage examples / tutorials can be distributed as simple links. -
55. Promises made to the user must be kept. +
56. Present clear and desirable goals to the user and offer an easy way for reaching them. -
57. Prevent fear of errors by allowing easy undo in all circumstances (using, installation etc.). -
58. React rapidly to negative feedback. Most users will not report negative feedback, but will just stop using DAPS. Negative feedback is invaluable in detecting problems. -
59. Easy and clearly instructed step-by-step installation. -
60. Uninstalling should be very easy. -
61. Easy and tutored step-by-step first usage experience. -
62. Provide help (or links to them) from within DAPS to prevent potential unavailability of external materials. (Integrate to course materials.) -
63. Do not force the user to answer questions (use defaults), but allow changing defaults if specifically desired. -
64. DAPS's ease of use should be tested with user tests. -

3.2.16 Implementation Status Chart

This chart summarizes the design requirements for DAPS and their current state of implementation. The notation is the same as in chapter 3.2 (- unimplemented, + partially implemented, ++ fully implemented).

Promote Intrinsic Motives

1. Dialog should promote improving skills and discourage to worry about grades. +
2. The focus should be on the proper procedure, not the end result. -
3. Metacognitive instruction and knowledge can be included. -
4. If student is frustrated due to assignment not progressing, DAPS can remind that the only reason for the assignment is to allow the student to learn. -

Effort Is Learning

5. DAPS never evaluates student skill, but needs to track student's effort. -
6. Track the time the student uses with each task. -
7. Time and effort correlate to learning and should be evaluated and rewarded. -
8. Rewarding can be as simple as complementing the student through dialog. +
9. After sufficient amount of time, remind the student not to overexert themselves. -

Provide Objective Assessment

10. Short quizzes before and after tests demonstrate progress to the student. -
11. Monitor student's overall time usage. -
12. Allow the student and course personnel to access this tracking information. -
13. Discourage speed competition. -

No Deadlines for Large of Difficult Tasks

14. Allow student time usage to be used as course requirements. -
15. Create a ready made CS1 package of assignments. -
16. A failed deadline, as any other severe failure, is a warning sign. -

Repetition Is Necessary

17. Do not automate or automatically assist any steps the student takes. +

Improve Memorization

18. Same cues must be repeated across all DAPS material. +
19. The used dictionary (cue-elaboration pairs) is easily and constantly accessible. -
20. The elaboration must be clear, single and distinct. -
21. The cue itself should be as distinct as possible. -

Repetition Must Be Successful

22. All presented CS1 goals must be reached. -
23. Each student should be monitored and intervened if failures start to accrue. -
24. Individual tasks should be monitored. Tasks with high failure rate should be fixed or removed. -

Easy Access to Relevant Information

25. All needed materials should be easily accessible (for example through hyperlinks). -
26. Good external sources of information can be linked or suggested. -

Prevent Avoidance

27. All tasks that are given to the student by DAPS, should also be completed. +
28. A long and hard effort should never result in failure. +

Eliminate Deadlocks

29. When a student gets stuck, there must always be a way forward. +
30. Remind the students, that if a problem bothers them and can not be solved even after several hours of trying, they should always contact course personnel for help before giving up and trying to forget it. -
31. “A panic button” in the interface to provide advice when all seems to fail. -

Avoid Dangerous Repetition

32. Identify harmful repetition and interrupt it in an unintrusive and considerate way. -

Maintain Domain of Gains

33. DAPS responses should be overly positive and encouraging. (But not silly.) +
34. DAPS responses should be honest. -
35. If negative outcome is clear, it must be handled. -
36. DAPS responses should not contain negative connotations. +

Reduce Extrinsic Cognitive Load

37. Instead of referring to a code line number (indirectly), refer to it with color change or arrow (direct reference). -
38. Prevent *split-attention* by keeping all DAPS information in DAPS dialog window. +
39. DAPS handles assisting through a large number of easy small steps. +

Metacognitive Instruction

40. DAPS eases cognitive load by facilitating schema construction. -
41. To divide resource costs uniformly, DAPS can encourage users to work every day and for reasonable hours at a time. -

Actively Correct Erroneous Perceptions

42. DAPS must provide easy way of changing previous incorrect perceptions. -
43. Old material must be easily accessible (previous exercises or explanations). -
44. Explanations must be easily accessible (hyperlinks). -
45. Previous steps must be easily accessible (undo/redo). -

Key / Threshold Concept Verification

46. In severe difficulties, tell that the cause might be in not understanding key concepts. -

Visualizations

47. Create or use external debugger to gain access to dynamic runtime information. +
48. The added visual information should be very fast to examine. -
49. The syntax of additional visual components must be intuitive, clearly textually explained and preferably instructed in tutorial. -
50. Add numbering or arrows into the code display to visualize program flow. -
51. Add variable data into the code display. -

Ease Third Party Adoption of DAPS

52. Adopting: Interface design must be consistent (inhibits cue-dependent forgetting). +
53. Adopting: User feedback should be encouraged. -
54. Adopting: Attractive presentation of features (outside Eclipse). -
55. Adopting: Promises made to the user must be kept. +
56. Adopting: Present clear and desirable goals to the user and offer an easy way for reaching them. -
57. Adopting: Prevent fear of errors by allowing easy undo in all circumstances (using, installation etc.). -
58. Adopting: React rapidly to negative feedback. -
59. Adopting: Easy and clearly instructed step-by-step installation. -
60. Adopting: Uninstalling should be very easy. -
61. Adopting: Easy and tutored step-by-step first usage experience. -
62. Adopting: Provide help (or links to them) from within DAPS to prevent potential unavailability of external materials. -
63. Adopting: Do not force users to answer questions (use defaults), but allow changes. -
64. Adopting: DAPS's ease of use should be tested with user tests. -

Table 1: DAPS implementation status chart

4 Eclipse DAPS – Specification and Implementation

4.1 Platform – Eclipse

For the project resources perspective, implementation and maintenance are the highest singular costs. Using an already made and maintained platform provides almost all required components and drastically reduces both implementation and maintenance costs. Eclipse's wide utilization in the programming community and industry provide a more familiar interface, and contribute to the ease of adopting the tool. In addition Eclipse ecosystem already contains many features that ease the adoption of new software, such as Eclipse's plugin architecture and automated over the network installation and updating features. Eclipse is also already used on many Aalto university CS courses.

Negative qualities include Eclipse's high complexity that increases extrinsic cognitive load by providing large amount of irrelevant features that split student's attention needlessly. The power provided by rapid development is not needed in CS1 and also results in many unfinished, confusing or conflicting details. While the students can benefit greatly later on, the high learning curve clearly presents a problematic challenge in the beginning of the studies. As Eclipse's modular design is also geared towards using Eclipse as a software platform and allows easy removal of features, further work could include using a feature stripped Eclipse, which could significantly reduce initial difficulties. However, even without these modifications, Eclipse presents a highly valuable and excellent platform for CS1 and DAPS.

4.2 Architecture

Currently DAPS has four main architectural parts:

- Integrated debugger, which analyzes the student's code to identify problems to which help can be provided.
- Student-Assistant Dialogue, which guides the student in learning and solving bugs.
- Additional help materials, which consists of linked textual, www, course etc. materials relevant to the task or student's interests.
- Eclipse integration, which packages DAPS into an Eclipse plugin and implements the user interface.

4.2.1 Integrated Debugger – Jython Python Interpreter

Integrated debugger is implemented through encapsulated and unexposed Jython module. Tight integration was chosen partially from necessity as obtaining detailed debugging information dives deep into compiler/interpreter implementation. High level of interconnectivity reduces the possibility of using exposed interfaces and increases side effect risks from updates to external Jython. While this protects DAPS from internal errors, it can potentially cause user perceivable inconsistencies in behavior, as the self contained Jython is decoupled from the compiler used in the actual compilation of the code (for example Python with PyDev interface). The side effects of this decision are therefore the increased maintenance cost and possible inconsistencies from slower update speed compared to using independent Jython.

4.2.1.1 Program Dynamics and Flow – The Line Numbers of Executed Code

One benefit from debugger, is the execution order of the program's lines. Many errors stem from inability to correctly predict the execution order, and within the restricted confines of CS1, program flow remains simple enough to be simply displayed. While debuggers or simple print commands can be used for much of the same benefits, debuggers can be dauntingly complex and debugging skills in general are poor.

4.2.2 Student-Assistant Dialogue – Assistant Instead of a Tool[5]

One of the most difficult tasks in providing the assistance is having to rely on a preprogrammed dialog. Only very few eventualities can be considered and much less can be implemented. While the language definition is finite, the mental processes of the student are not, and there is no limit to the different types of errors or misconception the student might make or have. This problem is partially solved by providing assistance on most frequent types of errors and remaining encouraging even when no solutions are in sight. However, many of the problems can be avoided by giving up on trying to fix the problems with infinite variation for the student. Instead, we focus on providing additional information, motivation and metacognitive skills that let the student solve the problems by themselves.

4.2.2.1 Unintrusive Initiation

The initiation of dialog should be unintrusive, but noticeable enough to be found. DAPS implement this by adding its icon to PyDev perspective's toolbar and promoting the icon at selected times, when the student could benefit from DAPS assistance. The promotions are small, visual and do not hide other interface components or alter the interface permanently. To add the view into PyDev perspective, the student must click the DAPS icon in the toolbar.

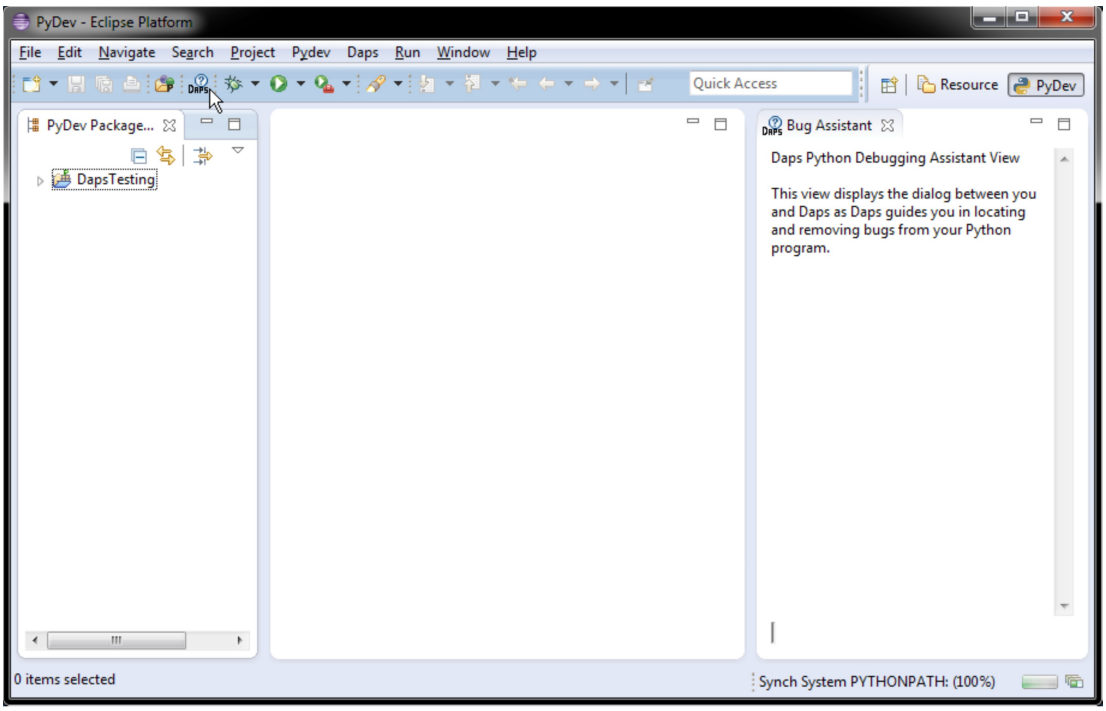

Illustration 4: DAPS dialog view in Eclipse PyDev perspective

[5] Most of the python code that is visible in pictures demonstrating DAPS was provided by Teemu Sirkiä (Aalto University, School of Science and Technology)

4.2.2.2 Minimize Negative Perceptions

The assistant never “takes over” the interface or displays pop-ups that grab focus, so ignoring it is easy and disruptions to the user's thought processes is minimal. Using DAPS always comes from the user's own initiative. If DAPS is perceived as bothersome or useless, the user can ignore or close the view at any time, regardless of the dialog or debugger state. This should also help to minimize the duration of any negative perception from DAPS. Even if DAPS is perceived useless for any given problem, the interaction can still be as pleasant as possible. The lack of animosity can increase the chances of DAPS being used again, and the “less productive” time is still spent on learning about the domain, even when the most pressing piece of information remains elusive.

4.2.2.3 Prevent Deadlocks

Additionally, as DAPS also aims to improve the user's metacognitive skills, the user's perception of uselessness does not necessarily mean, the user did not enjoy or benefit from the experience. The tool may simply provide a way for the user to regain motivation and self appreciation after battling with a difficult problem without success. When a problem can not fixed immediately DAPS tries to delay the direct confrontation, and meanwhile provide something else to do. While this does not further the immediate task, it can remove dissonance, reduce damage from previous failure and build motivational and information resources to carry on.

4.2.2.4 Provides Only Information – User Remains In Control

Assistance is provided in the form of information and advice. Nothing is done for the user. This reduces the amount of side-effects that might confuse the user, but more importantly the user needs these small easy tasks for amassing repetition. Here is a typical example:

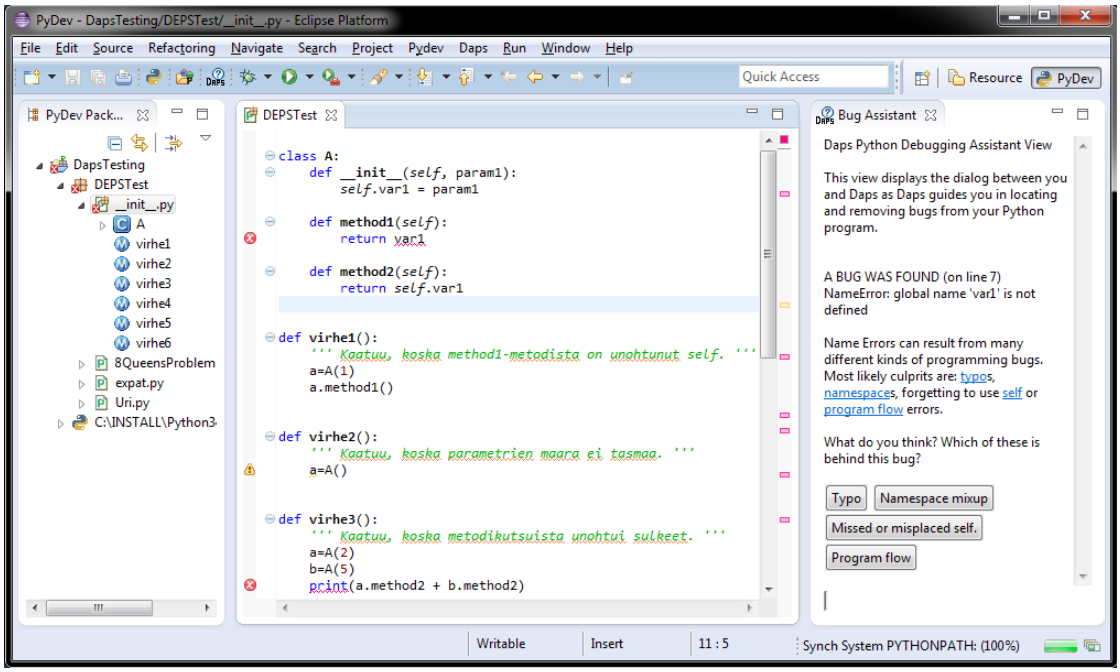

Illustration 5: DAPS provides easy tasks for the user

Here DAPS has found a bug using its debugger on the user's code. It gives clues to the source of the problem, and provides links to additional information on the possible source. DAPS attempts to steer the user's interest into programming concepts (*intrinsic motivation*) and away from completing the program (*instrumental motivation*). Figuring out the real reason behind the error is difficult or impossible for DAPS, but as DAPS only helps the user in exploring the error sources, the exact source does not matter. In this example, even when the source is the forgotten use of `self`, the user starts to explore the possibility of program flow error:

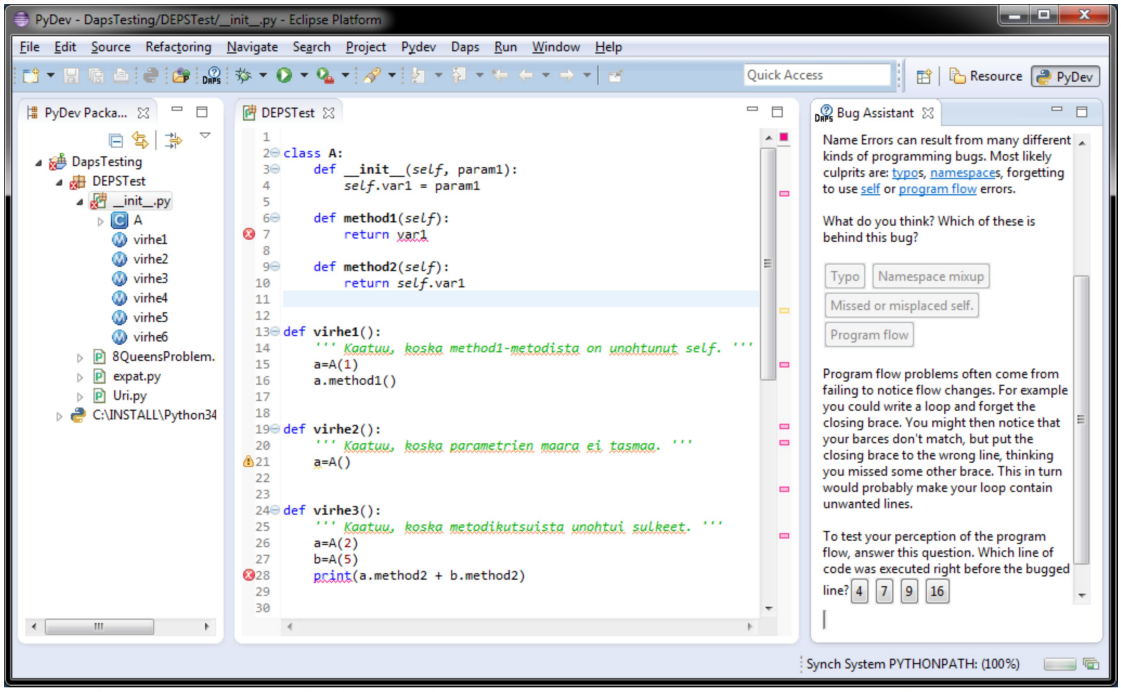

Illustration 6: DAPS tests the user's understanding of program flow

To explore the user's understanding of program flow, little general information and simple questions are used to engage the user before providing the information from the debugger. Still concrete benefits can be given (such as the execution order of program lines).

4.2.2.5 A Domain of Gains – Effort Guarantees Success

To provide repetitive success, the tasks and goals that are given to the user can be solved without too much work. This is implemented by asking multiple choice questions that always provide information needed for solving them. If the problem is understood, answering is trivial and takes little time, but still provides the vital repetition for important cues and terms. Where possible, randomization is used to prevent memorization and to enforce the semantic processing of the problem.

4.2.2.6 Overly Positive Framing

If the user makes an error in answering to a question, the response encourages to think a bit and try some other solution. Progress is rewarded by positive feedback, and failures are simply ignored. This is to create a domain of gains and to promote using new information and skills. This overly positive feedback might be perceived even silly, but this perception would actually be welcome, as relaxation improves memorization of new information. It would also help to offset any mental stress of persisting dissonance.

4.2.3 Additional Help Materials

Internet community surrounding Eclipse provides an enormous support for practically any conceivable usage problem. The difficulty here is that CS1 student's skills in navigating this endless sea of information are quite poor (at least in the domain of programming). As general knowledge structures of the domain are not understood, even answers to basic problems can be hard-pressed to find, increasing the cost for even the most basic interaction tasks, and unnecessarily hindering the student in obtaining coherent understanding and reducing dissonance. Assistant can alleviate some of these problems by providing a plethora of context dependent help materials. Used programming terms can be linked to explanations or guides to obtaining the information elsewhere. This not only provides help on demand, but introduces valuable information sources that can be utilized later on. The guiding principle here is to prevent dead-ends. In any state, there should always be a way forward, which is both encouraging and a reasonably "good" choice for the student to head for.

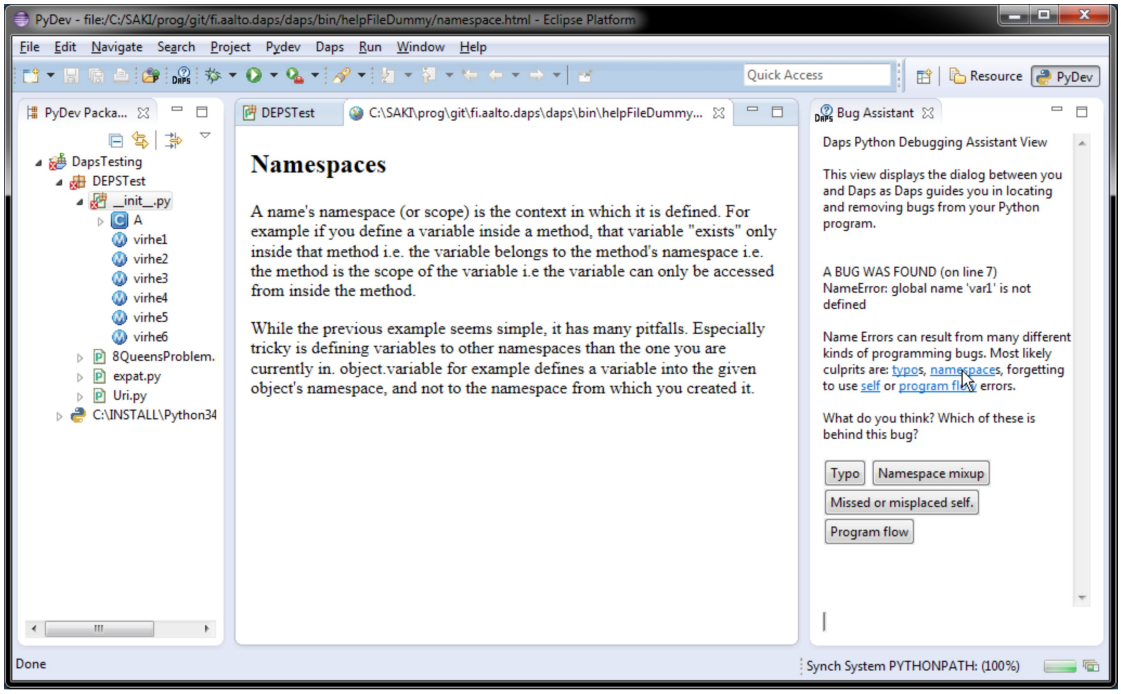

Illustration 7: DAPS provides easily accessible reference

There is no need to avoid redundancy in providing basic information, quite the opposite, as the repetition is the key to behavioral development. These simple tasks and the associated cues need to be fired again and again to make them dominant and to weed out the partially correct but still erroneous misconceptions used in the early stages of learning.

4.2.4 Eclipse Integration – Implementation Details

A view taken from the *DAPS javadoc* provides an overall picture of the current implementation. More detailed explanations are provided in the javadoc itself, as it is the only official and up-to-date documentation for DAPS.

Package	Description
daps	DAPS - Debugging Assistant for Programming Students
daps.actions	MVC controller implementations for this plugin's Eclipse GUI embedded controls.
daps.engine	Provides the MVC model.
daps.engine.debugger	Compiler and debugger classes and interfaces.
daps.engine.dialog	Contains the all dialog content classes and nothing else.
org.eclipse.wb.swt	
util	

Table 2: DAPS packages

Eclipse integration (Eclipse specific) code consists of all packages excluding daps.engine and those under it. These parts implement various hooks for Eclipse to start and interact with DAPS plugin. In particular daps.AssistantView implements most of the graphical UI of the DAPS's dialog view (the singular view that presents the interactive dialog to the user).

daps.engine is responsible from the model part of the model-view-controller design pattern. It is independent from Eclipse. The daps.engine.Coordinator controls the rest of the engine and is responsible of initialization and organization of the data structures and governs interaction between modules. As the environment is multithreaded, Coordinator also forms the root of a DAG (directed acyclic graph) that provides a strict hierarchy for synchronization and deadlock prevention by allowing direct function calls to only own and child functions. References to parents or other branches are WeakReferences (for proper garbage collection) and their functions can only be called through delayed execution.

daps.engine.dialog.AssistantDialog is the master class for the user interaction and currently provides a crude prototype dialog with which to collect and provide information. Dialog is responsible for most of the decision making, which will only increase with additional features such as user behavior analysis (to identify dissonance) and activity monitoring. In addition the actual tutoring scenarios and complex dialog trees will be implemented here. Dialog is also responsible for inserting hyperlinks and other reference tokens into the view, even when the visual implementation and rendering of these features rests on daps.AssistantView.

daps.engine.debugger.Debugger provides the debugger integration with Jython. Debugger collects information from the user written code, stores it and provides it for the dialog.

Eclipse stores much control data outside the classes and files such as MANIFEST.MF, plugin.xml and build.properties contain important information. Visual inspection of these files provides much information even when their modification is practically always done through GUI. Version control is currently done with Git.

In terms of future development and task management, the development environment contains a listing of future options in a text file that currently serves as the project's scheduler (DAPS_TODO.txt). As many design requirements are fairly general in nature and require to be taken into account when implementing any single feature, it is preferable to be familiar with this document and chapter 37 from *Handbook of Human-Computer Interaction* (Anderson et al. 1997). The recommended chapter also contains good guidance in implementing the

DAPS's currently most lacking feature (the dialog tree). Also the rule system needs a clear and extensible implementation, which allows easy addition of dialog material.

5 Comparison to Other Tools

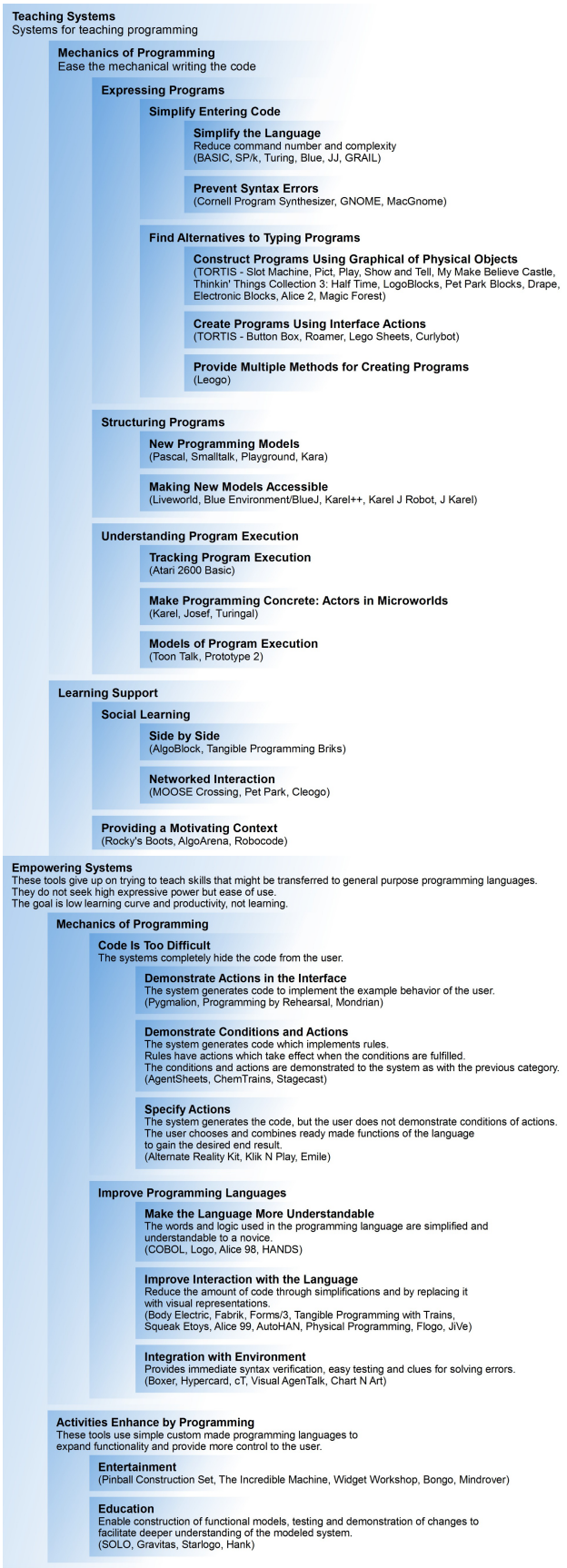

Illustration 8: Illustration is based on the taxonomy (Kelleher & Pausch 2005)

A taxonomy of programming environments and languages for novice programmers (Kelleher & Pausch 2005) provides a glimpse of the high number of tools and approaches used in making programming easier to learn. "*The systems are organized by their primary goal, either to teach programming or to use programming to empower their users, and then, by each system's authors' approach, to making learning to program easier for novice programmers.*"

To find the DAPS's peers to facilitate the comparison, DAPS's main features (some are still unimplemented) and qualities are needed (chapter 1.3):

- Main approach is to protect user from circumstances that cause *persisting cognitive dissonance*.
- Main interaction in the form of continual textual dialog.
- *Added value features* (debugger and visualizations) motivate to use DAPS and provide the necessary interaction for dissonance relieving elements.

In the taxonomy DAPS would most likely fall under *Teaching Systems → Learning Support → Providing a Motivating Context*. However the tools under that category are essentially games that incorporate programming elements or logic as a part of gameplay, and while the main approach for all tools is to affect the motivation, they bear little resemblance with DAPS. As DAPS shares qualities with several tools in different categories of the taxonomy the definition of DAPS's peer-tools is difficult.

The variation in features, supported work flows and target user groups pose a significant challenge in comparing the

available tools. If individual features are compared, already very simple tools (such as python's pdb module) allow setting breakpoints to access dynamic variable data, and the data can even be tracked without any tools by simply adding printing expressions into the code. An example of a more extensive freely available visual debugger could be Winpdb (Aides 2014).

IDEs (integrated development environments) naturally offer debugging facilities and there are many heavy weight commercial ides that contain visual debugging tools such as Komodo IDE (ActiveState 2014), IntelliJ IDEA (JetBrains 2104), PyCharm (JetBrains 2014) and Visual studio (Microsoft 2014), but are directed towards professional software development and hence offer more power instead of low learning curve. Still the chosen platform for DAPS is Eclipse IDE, which is equally directed towards professional software development.

However, there are also interesting examples of IDEs that are targeted towards teaching programming. BlueJ (Barnes & Kölling 2014) is a Java IDE to support the learning and teaching of object oriented programming (Kölling 2008). In contrast to previous IDE examples BlueJ is vastly simplified. It contains tools such as visual debugger, setting breakpoints, independent instance creation, access of variables through inspection and direct interface access to methods. DAPS also aims to contain similar debugging features, but in contrast to DAPS BlueJ does not provide dialog or other methods to instruct the student where to go next, but leaves those decisions to the user or course instruction.

If we consider the accessibility of information, tutorials such as the official python tutorial (Python Software Foundation 2014) or the learnpython.org's excellent tutorial with in-browser interpreter (LearnPython.org 2014) offer accessible language reference, but leave the resolution of encountered problems to the user. While tutorials are clearly far away from tools such as DAPS, the learning tasks have several distinct aspects and these aspects do not necessarily need to be presented by the same source. This also means that the lack of an important feature does not necessarily reduce the value of a tool, if such a feature can be concurrently provided by another source in the learning environment. For example, DAPS can ease referencing by inserting hyperlinks into its dialog, but if a suitable dictionary were open in a web browser, writing the name in a search field and clicking search would not be difficult. This dictionary could then be referenced even for terms DAPS had not hyperlinks associated. A more prominent problem would be the reduced linkage, which could result in terminological discrepancies, increase *extrinsic cognitive load* and hamper memorization by reduced repetition of *cues*, but there are also clear benefits in a modular learning environment which allows dynamic composition of multiple tools.

When we further include the conclusion, that the programming language is merely irrelevant syntax and the language independent features such as loops, functions and variables are the real learning tasks, the comparison can no longer be limited to any single language. As programming can be expressed through visual or even physical objects, the limitation to textual representation becomes more linked to the productivity requirements of professional programming than the CS1 level learning tasks. For example Lego NXT robots (Markham & King 2010) have been used in teaching CS1.

As the type of interaction is the most important property of the tools used (Hundhausen et al. 2002), it remains unclear how relevant any comparison of features can be, when taken out of teaching context that clearly defines the validity of each feature. In the end, the tool must viably support the specific user's learning process within the specific learning context. The lack of a vital component or its failed usage would probably be far more detrimental to

learning than the quality of all other components combined. This is also the conclusion hypothesized in chapter 2.1.

5.1 ITS Evaluation Is Difficult – User Testing Might be Better

The chosen context of DAPS, the Eclipse IDE with PyDev plugin, already contains many tools including debugging tools, which if understood and used, offer many superior features compared to DAPS (if tutoring specific features are excluded from the comparison). In fact the only “new” features that DAPS provides into Eclipse PyDev environment are the tutoring specific features, such as the dialog interface. The main function of DAPS is to provide direction for the user.

ITS systems are complex specialized systems requiring large development effort and as such are far fewer than general programming tools. They are also far more integrated to their environment and therefore less accessible. As a result their evaluation and comparison is laborious and difficult. While providing a far more informed and complete rational Anderson et al. note: *In short, the evaluation process is somewhat more complex than for most software environments* (Anderson et al. 1997).The conclusion taken here is that while familiarity with ITS systems is beneficial and features can be individually scrutinized for implementation in DAPS, such as using Bayesian networks for modeling the student's learning in BITS (*a web-based bayesian intelligent tutoring system for computer programming*) (Butz et al. 2006), the comparison of the whole tool should be based on empirical student learning results, and is deferred as future work after minimum viability is reached.

6 Summary and Future Work

As the effects of lack of skills and variation in ability (IQ) become negligible in the later stages of learning (chapter 2.4.1.2), all the students have "what it takes". Consequently the vicious cycle to quit develops based greatly on temporary or outright false perception of poor success. Even more disheartening is that this perception forms regardless of the course personnel's goal to help students learn and share their passion for computing. While much of the students' negative view seems out of place, still this bleak perception and commitment into dropping out is formed for 20-40 percent of the CS1 students (chapter 2.2).

Cognitive dissonance (chapter 2.2.3) is an established and well founded theory and explains well the phenomena of changing *cognitions* (thoughts) and *behaviors* in response to gaining new information from one's surroundings. It maintains our thoughts and actions *consonant* (consistent) and prevents conflicting and self defeating processes endangering functionality and in this way maintains productivity. The ever increasing discomfort of accumulating dissonance forces change even when the cost for the change is high and allows even difficult barriers to be overcome if the need is dire. Also the reasons given by dropped out CS1 students and their mounting discomfort suit well to the accumulation of dissonance (chapter 2.2.3.3).

While dissonance creates a very powerful motivator, it is harmful in the context of complex learning tasks. Dissonance drains *working memory capacity* and reduces performance in working memory intensive tasks (chapter 2.3.1.4). While simple tasks benefit from the additional motivational momentum, in the context of CS1's high *intrinsic cognitive load*, the students who are faced with problems find their performance degrade when it is most needed. The high interconnectivity of CS1 will also prevent pushing through by *avoiding* the unsolved problems, a solution that would provide momentary relief in many other contexts. These harmful feedbacks can result in an entrapment by escalating dissonance, where dissonance pushes the student ever harder forward, but in the same time drains the very means needed for the progress. If this continues too long and the dissonance rises too high, avoiding it becomes mandatory (chapter 2.2.3.7). Unfortunately the student's only way to avoid the interconnected sources of the dissonance is to avoid the whole interconnected domain, hence leading to the abandonment of the studies.

Also other factors significantly contribute to this negative outcome. While CS1 skills can be mastered by all students the different backgrounds (and differing programming related skills, especially the understanding of abstract entities such as *threshold concepts*) mean that the required workload can vary considerably between students, making the course demands simply impossible for some of them. This discrepancy is not likely to be understood or detected by the students and in combination to the following successive failures, leads to unfair, harsh and incorrectly negative self evaluation, partly due to poor *metacognitive skills* (chapter 2.4.1) such as planning and self knowledge. In addition the poor metacognitive skills combined to fixed outside constraints can lead to resource depletion leaving the student defenseless against unforeseen future difficulties. As a result, the perception of the course changes into a *domain of losses* (chapter 2.2.4), which derails the student from proper process, encourages unwise risk taking and short term gain over long term investment and further damages learning and productivity.

Rapid intervention and preferably preventative measures are needed, as the negative feedbacks of *resource depletion*, *excessive cognitive load* and *avoidance failure* (chapter 3.1) will soon galvanize the students against any attempts that are perceived to prevent avoidance or other forms of dissonance reduction. At this point intervention will be very difficult as the fear of additional dissonance prevents the students from committing, and dissonance reduction enforces course personnel to be viewed in negative light (chapter 2.2.3.2).

Still there are ways for prevention and intervention. Maintaining *domain on gains*, using *positive framing*, instruction in *metacognitive skills* and promotion of *intrinsic motivation* are some of the preventive methods. Different monitoring methods might also provide early enough opportunity for intervention, when the student is still seeking help in succeeding, instead of seeking help in avoiding. As the student's perception of own performance is a major factor, sufficient intervention might be as little as few encouraging words, if the intervention is done early enough. While social interaction is far stronger source for this encouragement, DAPS has the potential of proving it more frequently.

The time consumption in terms of course resource expenditure is a major benefit for automated tutoring systems such as DAPS. With the tool the normally limited time consumption becomes near irrelevant, need for rapid progress is relaxed and the domain of gains can be maintained with constant stream of easy tasks, providing easy gratification and perception of progress and success. In these surroundings the needed *repetition* for memorization and behavioral changes (i.e. learning) becomes easily reachable. As memorization benefits little or none from the intent of memorization, the circumstances for this time intensive task can be made as pleasant as possible.

6.1 Limitations

The author has no experience in running a class and only a very modest amount of experience in teaching a small group (3-6) of students. This kind of experience would undoubtedly be useful or even vital.

6.2 Recommendations, Current state of DAPS and Future Work

The harmful interaction between *high cognitive load* and *persisting cognitive dissonance* may have been overlooked previously and should warrant more detailed study. Techniques such as instruction on *metacognitive skills*, *positive framing* and focus on *intrinsic motivation* should provide benefits beyond reducing CS1 drop-out rates and should contain little risk. Progress monitoring could help in identifying students that benefit from additional help in dissonance reduction.

DAPS is a proof of concept still at prototype stage, and requires several features before it could be usefully utilized in a CS1 environment. However, only a fraction of the many suggested features are required for viability, allowing DAPS to become a practical CS1 assistance tool with a reasonable amount of future work. While DAPS has not yet reached utility, this work has also provided a novel perspective of *persisting dissonance behind CS1 drop-out rates* (chapter 3.1) and followed with several concrete strategies that can be immediately employed to contain the harmful effects of persisting dissonance.

7 References

Ackerman, P., 1987. Individual differences in skill learning: An integration of psychometric and information processing perspectives. *Psychological bulletin*, 102(1), pp.3–27. Available at: http://psycnet.apa.org/journals/bul/102/1/3/ [Accessed July 23, 2014].

ActiveState, 2014. Komodo IDE - Cross-Platform IDE for all your major languages. Available at: http://komodoide.com/ [Accessed December 1, 2014].

Aides, N., 2014. Winpdb - A Platform Independent Python Debugger. Available at: http://winpdb.org/ [Accessed December 1, 2014].

Anderson, J.R., Cobertt, A.T. & Koedinger, K.R., 1997. Intelligent Tutoring Systems. In M. G. Helander, T. K. Landauer, & P. Prabhu, eds. *Handbook of Human-Computer Interaction*. Amsterdam: Elsevier Science.

Anderson, L.W. & Krathwohl, D.R., 2001. *A Taxonomy for Learning, Teaching, and Assessing, A Revision of Bloom's Taxonomy of Educational Objectives*,

Bandura, A., 1989. Self-regulation of motivation and action through internal standards and goal systems. In *Goal concepts in personality and social psychology*. pp. 19–85.

Barnes, D. & Kölling, M., 2014. BlueJ - A free Java Development Environment designed for beginners, used by millions worldwide. Available at: http://bluej.org/ [Accessed December 3, 2014].

Ben-Ari, M., 2001. Constructivism in computer science education. *Journal of Computers in Mathematics and Science Teaching*, 20(1), pp.45–73. Available at: http://dl.acm.org/citation.cfm?id=274308 [Accessed September 25, 2014].

Bloom, B.S. et al., 1956. *Taxonomy of Educational Objectives, The Classification of Educational Goals: Handbook I: Cognitive Domain*, New York: David McKay.

Borasi, R., 1996. *Reconceiving Mathematics Instruction: A Focus on Errors*, Norwood, NJ: Ablex Pub.

Butz, C.J., Hua, S. & Maguire, R.B., 2006. A Web-based Bayesian Intelligent Tutoring System for Computer Programming. *Web Intelli. and Agent Sys.*, 4(1), pp.77–97. Available at: http://dl.acm.org/citation.cfm?id=1239784.1239789.

Carpenter, T. & Lehrer, R., 1999. Teaching and learning mathematics with understanding. *Mathematics classrooms that promote understanding*, pp.22–33. Available at: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Teaching+and+Learning+Mathematics+With+Understanding#0 [Accessed October 26, 2013].

Chase, W.G. & Simon, H.A., 1973. Perception in chess. *Cognitive Psychology*, 4(1), pp.55–81.

Coleman, J., 1968. Equality of educational opportunity. *Integrated Education*. Available at: http://www.tandfonline.com/doi/pdf/10.1080/0020486680060504 [Accessed October 26, 2013].

Craik, F.I.M. & Lockhart, R.S., 1972. Levels of Processing : A Framework for Memory Research 1. *Journal of Verbal Learning and Verbal Behavior*, 684, pp.671–684.

Dollard, J. & Miller, N.E., 1950. *Personality And Psychotherapy*, New York: McGraw-Hill.

Engle, R.W. et al., 1999. Working memory, short-term memory, and general fluid intelligence: a latent-variable approach. *Journal of Experimental Psychology: General*, 128(3), pp.309–331. Available at: http://psycnet.apa.org/psycinfo/1999-11137-004 [Accessed August 8, 2014].

Eysenck, M.W. & Keane, M.T., 2005. *Cognitive Psychology : a Student's Handbook* 5th ed., Psychology Press Ltd.

Festinger, L., 1957. *A Theory of Cognitive Dissonance*, Stanford, California: Stanford University Press.

Fitzgerald, S. & Lewandowski, G., 2008. Debugging: finding, fixing and flailing, a multi-institutional study of novice debuggers. *Computer Science Education*, 18(2), pp.93–116. Available at: http://www.tandfonline.com/doi/abs/10.1080/08993400802114508 [Accessed October 27, 2013].

Gerrig, R.J. & Zimbardo, P.G., 2002. American Psychological Association - Glossary of Psychological Terms. *Psychology And Life*. Available at: http://www.apa.org/research/action/glossary.aspx [Accessed February 6, 2014].

Ginat, D., 2013. Constructive use of errors in teaching CS1. *ACM SIGCSE Bulletin*, p.353.

Gupta, P. & Cohen, N.J., 2002. Theoretical and computational analysis of skill learning, repetition priming, and procedural memory. *Psychological Review*, 109(2), pp.401–448. Available at: http://doi.apa.org/getdoi.cfm?doi=10.1037/0033-295X.109.2.401 [Accessed July 15, 2014].

Hundhausen, C., Douglas, S. & Stasko, J., 2002. A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages & Computing*, 13(3), pp.259–290. Available at: http://www.sciencedirect.com/science/article/pii/S1045926X02902375 [Accessed October 26, 2013].

Hyde, T.S. & Jenkins, J.J., 1973. Recall for words as a function of semantic, graphic and syntactic orienting tasks. *Journal of Verbal Learning & Verbal Behavior*, 12, pp.471–480.

JetBrains, 2104. IntelliJ IDEA - The Most Intelligent Java IDE. Available at: https://www.jetbrains.com/idea/ [Accessed December 1, 2014].

JetBrains, 2014. PyCharm - The Most Intelligent Python IDE. Available at: https://www.jetbrains.com/pycharm/ [Accessed December 1, 2014].

Jonas, E. et al., 2001. Confirmation bias in sequential information search after preliminary decisions: an expansion of dissonance theoretical research on selective exposure to information. *Journal of personality and social psychology*, 80(4), pp.557–71. Available at: http://www.ncbi.nlm.nih.gov/pubmed/11316221.

Kahneman, D. & Tversky, A., 1984. Choices, values, and frames. *American Psychologist*, 39(4), pp.341–350. Available at: http://content.apa.org/journals/amp/39/4/341 [Accessed July 21, 2014].

Kahneman, D. & Tversky, A., 1979. Prospect theory: An analysis of decision under risk. *Econometrica: Journal of the Econometric Society*, 47(2), pp.263–292. Available at: http://www.jstor.org/stable/1914185 [Accessed July 21, 2014].

Kelleher, C. & Pausch, R., 2005. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys (CSUR)*, 37(2), pp.83–137. Available at: http://dl.acm.org/citation.cfm?id=1089734 [Accessed October 26, 2013].

Kelly, G.A., 1958. Man's construction of his alternatives. In G. Lindzey, ed. *Assessment of Human Motives*. New York: Holt, Rinehart and Winston, pp. 33–64.

Kinnunen, P. & Malmi, L., 2006. Why students drop out CS1 course? *Computer Science Education (2006)*, 24(7), pp.97–108. Available at: http://dl.acm.org/citation.cfm?id=1151604 [Accessed April 2, 2014].

Kölling, M., 2008. Using bluej to introduce programming. In and M. K. J. Bennedsen, M. E. Caspersen, ed. *Reflections on the Teaching of Programming*. pp. 98–115.

Lamberts, K. & Goldstone, R.L. eds., 2005. *The Handbook of Cognition*, SAGE Publications Ltd.

LearnPython.org, 2014. LearnPython.org interactive Python tutorial. Available at: http://www.learnpython.org/ [Accessed December 1, 2014].

Markham, S. & King, K., 2010. Using personal robots in CS1: experiences, outcomes, and attitudinal influences. *ITiCSE'10 - Proceedings of the 2010 ACM SIGCSE Annual Conference on Innovation and Technology in Computer Science Education (2010)*, pp.204–208. Available at: http://dl.acm.org/citation.cfm?id=1822148 [Accessed December 2, 2014].

Marks, D.F., 1997. Biographical Dictionary of Psychology. In *Biographical Dictionary of Psychology*.

Martinie, M., Olive, T. & Milland, L., 2010. Cognitive dissonance induced by writing a counterattitudinal essay facilitates performance on simple tasks but not on complex tasks that involve working memory. *Journal of experimental social psychology*, 33(0), pp.1–14. Available at: http://www.sciencedirect.com/science/article/pii/S0022103109002716 [Accessed August 8, 2014].

Meyer, J. & Land, R., 2006. *Overcoming Barriers to Student Understanding: Threshold Concepts and Troublesome Knowledge*, Routledge.

Microsoft, 2014. Visual Studio. Available at: http://www.visualstudio.com/ [Accessed December 1, 2014].

Miller, G.A., 1994. The magical number seven, plus or minus two: some limits on our capacity for processing information. 1956. *Psychological review*, 101(2), pp.343–352.

Miller, N.E., 1944. Experimental studies of conflict behavior. In J. M. Hunt, ed. *Personality and behavior disorders*. New York: Ronald Press, pp. 431–465.

Ohlsson, S., 1996. 1996-Learning_from_Performance_Errors.pdf. *American Psychological Association*, 103(2), pp.241–262.

Paivio, A., 1971. *Imagery and Verbal Processes*, New York, Holt, Rinehart and Winston.

Pavlov, I.P., 1927. Conditioned Reflexes: An Investigation of the Physiological Activity of the Cerebral Cortex. Available at: http://psychclassics.yorku.ca/Pavlov/ [Accessed December 11, 2013].

Pervin, L.A., 2003. *The Science of Personality* 2nd ed., New York: Oxford University Press.

Poldrack, R. a & Gabrieli, J.D., 2001. Characterizing the neural mechanisms of skill learning and repetition priming: evidence from mirror reading. *Brain : a journal of neurology*, 124(Pt 1), pp.67–82. Available at: http://www.ncbi.nlm.nih.gov/pubmed/11133788.

Poldrack, R. & Selco, S., 1999. The relationship between skill learning and repetition priming: Experimental and computational analyses. *Journal of experimental psychology. Learning, memory, and cognition*, 25(1), pp.208–235. Available at: http://psycnet.apa.org/journals/xlm/25/1/208/ [Accessed July 15, 2014].

Python Software Foundation, 2014. The Official Python Tutorial. Available at: https://docs.python.org/3/tutorial/index.html [Accessed December 1, 2014].

Roll, I. et al., 2011. Improving students' help-seeking skills using metacognitive feedback in an intelligent tutoring system. *Learning and Instruction*, 21(2), pp.267–280. Available at: http://linkinghub.elsevier.com/retrieve/pii/S0959475210000538 [Accessed July 14, 2014].

Roussel, P., Elliot, A.J. & Feltman, R., 2011. The influence of achievement goals and social goals on help-seeking from peers in an academic context. *Learning and Instruction*, 21(3), pp.394–402. Available at: http://linkinghub.elsevier.com/retrieve/pii/S0959475210000447 [Accessed July 30, 2014].

Schraw, G., 1998. Promoting general metacognitive awareness. *Instructional science*, 26(1-2), pp.113–125. Available at: http://link.springer.com/article/10.1023/A:1003044231033 [Accessed July 22, 2014].

Secada, W.G. & Marrett, C.O.R.A.B., 2000. The Organizational Context of Teaching and Learning Changing Theoretical Perspectives. In *Handbook of the Sociology of Education*. pp. 37–64.

Sorva, J., 2012. *Visual Program Simulation in Introductory Programming Education*. Aalto University. Available at: https://aaltodoc.aalto.fi/handle/123456789/3534.

Swanson, H.L., 1990. Influence of metacognitive knowledge and aptitude on problem solving. *Journal of Educational Psychology*, 82(2), pp.306–314. Available at: http://doi.apa.org/getdoi.cfm?doi=10.1037/0022-0663.82.2.306.

Sweller, J., 1988. Cognitive load during problem solving: Effects on learning. *Cognitive science*, 12(2), pp.257–285. Available at: http://doi.wiley.com/10.1016/0364-0213(88)90023-7 [Accessed July 17, 2014].

Sweller, J., 1994. Cognitive load theory, learning difficulty, and instructional design. *Learning and instruction*, 4, pp.295–312. Available at: http://www.sciencedirect.com/science/article/pii/0959475294900035 [Accessed July 17, 2014].

Thomas, A. & Millar, P., 2012. Reducing the framing effect in older and younger adults by encouraging analytic processing. *The journals of gerontology. Series B, Psychological sciences and social sciences*, 67B(2), pp.139–149. Available at: http://http//psychsocgerontology.oxfordjournals.org/content/67B/2/139 [Accessed August 19, 2014].

Thomas, N.J.T., 2013. Mental Imagery - (Dual Coding and Common Coding Theories of Memory). In E. N. Zalta, ed. *The Stanford Encyclopedia of Philosophy.*

Watkins, O.C. & Watkins, M.J., 1975. Buildup of proactive inhibition as a cue-overload effect. *Journal of Experimental Psychology Human Learning and Memory*, 1(4), pp.442–452.

Watson, C. & Li, F.W.B., 2014. Failure rates in introductory programming revisited. *Proceedings of the 2014 conference on Innovation & technology in computer science education - ITiCSE '14*, pp.39–44. Available at: http://dl.acm.org/citation.cfm?doid=2591708.2591749.

Veen, V. van et al., 2009. Neural activity predicts attitude change in cognitive dissonance. *Nature Neuroscience*, 12(11).

Wrzesniewski, A. et al., 2014. Multiple types of motives don ' t multiply the motivation of West Point cadets. In *Proceedings of the National Academy of Sciences*. pp. 1–6.