Универзитет „Гоце Делчев“ - Штип

Факултет за информатика

Магистерски труд на тема:

Примена на Канбан методологијата и нејзините ефекти кај тимовите за развој на софтвер

Ментор:

Проф. д-р Сашо Коцески

Изработил:

Невенка Кировска

Број на индекс: 21081

Штип, 2015

Содржина:

Апстракт

Во секојдневниот живот се среќаваме и користиме голем број на уреди кои, покрај нивната физичка компонента, неминовно имаат и софтверска компонента. Овие уреди се најразлични, од персонални компјутери, мобилни телефони, таблети па сè до најразлични производствени машини. Развивањето на еден софтверски продукт не претставува само пишување на код и негово одржување. Овде станува збор за еден структуриран и планиран процес кој опфаќа повеќе фази. За реализирање на овој циклус на софтверскиот продукт постојат повеќе методи и техники, секоја со свои предности и недостатоци за даден конкретен проект. Овие методи неминовно се менуваат со цел подобрување на ефикасноста и ефективноста на самиот процес за развој на софтвер. Поаѓајќи од традиционалните модели, па сè до агилните методи за развој на софтвер, увидени се потребите за сè поподробно истражување и подобрување на овие методи со оглед на нивното влијание врз целокупната успешност на развојот на еден софтверски продукт.

Канбан претставува нова техника за управување, односно контрола на процесот за развој на софтвер. Првенствено, оваа техника најпрвин се употребила во Јапонија и тоа во производството. Оваа методологија применета во областа на софтверски развој е сè уште млада, меѓутоа, во светски рамки веќе е докажана како успешна практика со низа успешни софтверски проекти.

Целта на магистерската тема е претставување на Канбан методологијата и нејзина практична примена во процесот на развој и управување со софтверските проекти преку креирање на Канбан веб апликација. Популарноста на веб апликациите доаѓа од многубројните предности кои тие ги нудат. Една од најголемите предности е тоа што овозможуваат интеракција на луѓето без потреба за физичка средба, а ова е особено значајно кога самите лица кои работат на некој проект се географски дистрибуирани на големи растојанија. Друга значајна предност на веб апликацијата е тоа што не е потребно нејзино инсталирање локално на компјутерот. Членовите од тимот за развој на софтвер и пристапуваат од каде било и во кое било време. Сите податоци од апликацијата се наоѓаат онлајн и им се достапни на сите лица од тимот. Оваа Канбан веб апликација ќе се тестира и искористи практично од страна на

конкретен тим за развој на софтвер за даден софтверски проект. Резултатите од истата ќе се евалуираат и ќе се споредуваат со оние од останатите методологии за развој на софтвер. Целта е да се прикаже нејзината ефикасноста, ефективност, заштедата на време и ресурси итн.

Abstract

In our everyday life we encounter and use a range of devices that despite of their physical component also have a software component. These devices are all different varying from personal computers, mobile phones, tablets to various production machines. The development of one software product is not just writing code and take care of its maintenance. This is about a structured and planned process involving several stages. For realization of the cycled software product there are several methods and techniques, each with its own advantages and disadvantages for a particular project. These methods are changing inevitably in order to improve the efficiency and effectiveness of the software development process. Starting from the traditional models to the agile methods of software development, having in mind of their overall success of the software product development, different demands for detailed research and improvement were encountered.

Kanban is a new technique for management respectively a control of the software development process. Primarily this technique is first used in Japan and in a production environment. This methodology applied in the field of software development is still young, but in world frames it is already proven as a successful practice with a series of successful software products.

The purpose of the thesis is to present the Kanban methodology and its practical application in the development and management of software projects by creating a Kanban web application. The popularity of web applications comes from the many advantages they offer. One of the biggest advantages is that it enables human interactions without the need for a physical meeting and this is particularly important when the people working on a project are geographically far from each other. Another important advantage of a web application is that we don't have to install it locally on a computer. The members of one software development team can access the application from anywhere at any time. All data application are online and the team

users can access it. This Kanban application will be practically used and tested by a software development team for a given software product. The results of this usage will be evaluated and compared with the results of using other methodologies for software development. The goal is to show its effectiveness, efficiency, the saving of time and resources and so on.

1. Вовед

Во денешната динамичка бизнис и технолошка околина, организациите постојано ги прилагодуваат нивните структури, стратегии и политики како резултат на новите побарувања. Со зголемување на продуктивноста се намалуваат трошоците за извршување на операциите и се намалува времето за испорака. Едно од прашањата кои неминовно се поставува овде е: Како да развиеме подобар и поевтин софтвер и да го испорачаме на време за да ги задоволиме променливите кориснички барања. За постигнување на овие цели постојат неколку пристапи. Еден од пристапите се агилните методи (Agile methods - AM) кои се многу популарни во софтверската индустрија. Овие методи се покажаа како успешни решенија за проблемите кои беше тешко да се решат со традиционалните пристапи за софтверски развој. Lean пристапот произлегува од lean производството. Овој пристап најпрвин е развиен и успешно реализиран во автомобилската индустрија [22]. Во текот на изминатата декада, Lean пристапот[1] имаше особено влијание врз конкуренцијата на компаниите преку подобрување на ефикасноста на процесите и намалување на операцискиот губиток. Денес овој пристап се користи кај голем дел од компаниите на глобално ниво вклучувајќи ја и софтверската индустрија. Во софтверската индустрија станува збор за пристап кој многу брзо се развива, но, според студијата на Ahmad et al. [25], постои недостаток на научни истражувања за имплементацијата на Канбан во софтверскиот развој. Главната карактеристика, а и воедно и клучен фактор за популарноста на овој пристап е елиминацијата на сите видови губиток од развојниот процес. За да се постигне оваа цел авторите на една од најпознатите книги за овој концепт [22] предложиле низа на принципи[2]. Како аналогија во рамки на софтверскиот развој за губиток се смета

[1] Подетално опишан во секцијата 3.1
[2] Принципите се опишани во секцијата 3.1

сè она што не дава вредност за корисникот. Тука припаѓаат: нецелосно завршената работа, дополнителните процеси и особини, промената на работните задачи и дефектите [22].

Еден од начините на извршување на овој Lean концепт е Канбан кој веќе една декада успешно се користи во производствената компанија Тојота [17]. Почеста е и употребата на Канбан како додаток на Lean концептот и агилниот софтверски развој. Она што е од особено значење е сè поголемата популарност на Канбан во развојот на софтвер. Од аспект на развој на софтвер, почетоците на Канбан датираат од David J. Anderson [28] кој е главниот виновник за софтверска Канбан промоција. Во 2004 година Anderson [28] беше повикан во еден мал ИТ тим на Microsoft за да му помогне на тимот за визуелизација на работниот тек и ограничување на работата во тек. Целта за претставување на видливоста и ограничувањето на работата во тек е во контекст на идентификување на ограничувањата на даден процес со акцент на една задача во даден момент. Оваа техника го поддржува pull[3] пристапот. За разлика од овој пристап, во традиционалниот развој на софтвер се користи push[4] пристапот. Кај овој традиционален пристап работните задачи се „втурнуваат“ кон инженерите со инструкции за побрзо извршување на што е можно поголем број од задачите. Овој развој на софтвер потсетува на еден синџир каде работната задача од еден член на тимот се дава на друг на пример од програмер на тестер. Ова предизвикува застои во целокупниот процес кога некој од членовите е претрупан со многу работа или пак има проблем со работната задача. Она што Канбан го прави во овој случај е токму алтернативен начин на традиционалниот. Наместо „втурнување“ на работни задачи, истите се „повлекуваат“, а со тоа секој член од тимот работи на една задача во тековен момент. Кога доделената задача е завршена, членот од тимот автоматски ја „повлекува“ наредната задача.

Според студијата на Ahmad et al. [26], спроведена за подобрување на конкурентната позиција на финскиот софтвер во однос на глобалниот пазар, развиени се нови бизнис cloud модели, Lean софтверски модел и отворена софтверска cloud инфраструктура. Причина за нивната студија е недостатокот

[3] Подетално опишан во секцијата 3.4
[4] Подетално опишан во секцијата 3.4

од вакви студии за употребата на Канбан и неговата практична примена во софтверскиот развој [27] . Овде тие направиле анкета и неколку интервјуа кои во суштина ги имаат следните прашања:

- Зошто се користи Канбан во софтверските компании?
- Кои се предностите од користењето на Канбан?
- Со кои предизвици се соочуваме при адаптацијата на Канбан и кои се соодветните решенија?

Резултатите од студијата покажаа дека најголеми мотивациони фактори за адаптација на Канбан се подобрување на комуникацијата во тимот, намалување на времето за развој, зголемување на продуктивноста итн. Помеѓу предностите се вбројуваат подобра видливост, зголемена транспарентност и контрола на текот, а овие предности се слични на предностите кои и порано се среќаваат во литературата [28][29] како резултат од истражувања. Во однос на предизвиците со кои се соочуваа членовите од тимот, овде може да се стави акцент на недоволното искуство и обука за Канбан, што повлекува и слаба мотивација на тимот за обука на Канбан методот. Ова недоволно познавање на методот беше и причина за префрлување на работата на тимот кон стариот веќе познат начин на работа.

„In a company, not all business lines and top level management are familiar with Kanban. There are a few people who have knowledge but they also like to build their confidence to use the new Kanban approach. So, there is much resistance to change. The company lacks awareness about the existing mind-set issue. For example, for us, the release cycle is quite big; we are dealing with a huge requirement. In such a scenario, we need confidence that this can be done and delivered efficiently with this new Kanban thing. Nobody is willing to take the risk and start doing things the Kanban way. Everyone in the company knows that if we lose one release, we are out of the market, which means there's no company. The risk level is clear. It makes you feel safer to stay with what you already have“[26].

Како што покажува и заклучокот од самата студија [26], а и ќе биде детално опишано во поглавјата што следат, примената на Канбан софтверските решенија подразбира голем број на мотивациони фактори и придобивки, но и

голем број предизвици како резултат на недоволното искуство и сè уште младиот развој на овој методот.

2. Методи за развој на софтвер

Во софтверското инженерство методологијата за развој на софтвер, односно SDM[5], претставува рамка за структурирање, планирање и контрола на процесот за развивање на еден информационен систем.

2.1 Модел на водопад

Овој модел настанал во 70-тите години на 20-тиот век. Софтверскиот развој се реализира како низа на временски одвоени активности, почнувајќи од анализата на системските барања, па сè до финализирање на продуктот и негово одржување. Прогресот „тече“ од едно ниво кон друго, слично како еден водопад, од каде и доаѓа името на овој модел.

Овој модел ги има следните фази:

Анализа на барањата – Првиот чекор од овој модел е најзначаен бидејќи во овој чекор се собираат барања и информации за потребите на корисниците и се дефинираат. Во анализата се вклучуваат и информации од бизнис околината како и ограничувањата на системот, функциите кои тој треба да ги извршува, другите системи со кои тој треба да е компатибилен итн. За прибирање на овие и слични податоци се користат разни техники како што се интервјуа со корисиниците, дијаграми на кориснички случаи на употреба и друго. Краен документ од оваа фаза е документ за формална спецификација кој претставува влез во наредната фаза од овој модел.

Дизајн – Фазата на дизајн се состои од дефинирање на хардверската и софтверската архитектура, специфицирање на параметрите за безбедноста и перформансите, компонентите, модулите, интерфејсот и податоците, потоа избирање на околината за развој како и програмскиот јазик. Сето ова е со цел задоволување на наведените барања. Излезот од оваа фаза е спецификација/и

[5] Software Development Methodology

за дизајнот на системот кој, пак, од друга страна, се користи како влез во фазата на имплементација.

Имплементација – Оваа фаза се извршува од страна на тим на девелопери кој се состои од програмери, дизајнери на интерфејси, лица кои вршат тестирања итн. Тие користат разни алатки како што се компајлери, дебагери, интерпретери, разни едитори итн. Излезот од оваа фаза се имплементирани компоненти на софтверскиот продукт според претходно дефинираните барања.

Тестирање – После завршување на претходната фаза (имплементацијата) следува фазата на тестирање. Постојат различни начини и методи за откривање на багови (логички грешки во кодирањето) кои се појавиле во некоја од претходните фази. Постојат бројни алатки и методи наменети за оваа цел.

Одржување – Во повеќето случаи, изработениот софтвер кој е доставен до корисниците постои големи шанси дека повторно ќе претрпи промени. За таквите промени постојат голем број причини. Промена ќе се случи, на пример, при непредвиден влез во системот. На овој начин промените во системот директно влијаат врз софтверските операции. Софтверот треба да се приспособува на промените што можат да се случат за време на постразвојниот период.

Иако овој модел е еден од најстарите модели, сè уште се користи во пракса и тоа во случаи кога имаме големи системи каде барањата се концизни и јасни.

Слика бр. 1 – Модел на водопад

2.2 Агилни методи за развој на софтвер

2.2.1 Почетоци на развојот на агилните методи

Од 11-13 февруари 2001 г. во едно зимско одморалиште во Јута, се состанале група од 17 експерти наречена *The Agile Alliance* од областа на софтверското инженерство за да дискутираат за недостатоците и проблемите на тековните методи и методологии во софтверското инженерство. Токму од оваа средба произлезе документ наречен *Agile Manifesto* [1]. Овој документ ги содржи основите на агилните методи и од него произлегуваат следните значајни четири пораки и тоа:

1. *Индивидуалистите и интеракцијата* над процесите и алатките;
2. *Софтверот кој работи* над големата документација;
3. *Соработката со клиентот* над спогодбата за договорот и
4. *Одговорот на промените* над следењето на планот.

Овој документ се базира на следните 12 принципи [1]:

1. Нашиот најголем приоритет е да ги задоволиме потребите на корисниците преку рана и редовна испорака на квалитетен софтвер;
2. Промените во барањата се добредојдени, дури и во фазата на развој. Агилните процеси поттикнуваат промени за компетитивна предност на клиентите;
3. Испораката на софтвер треба да се реализира често, од неколку недели до неколку месеци;
4. Луѓето од бизнис секторот треба секојдневно да работат со девелоперите во текот на проектот;
5. Управувањето со проектите треба да го водат мотивирани индивидуалисти, откако претходно ќе им се овозможат сите потребни услови за работа;
6. Најефикасниот и ефективниот начин за пренесување на информации кон и во еден тим за развој е преку комуникација лице в лице;
7. Основна мерка за напредок е софтверот кој работи;
8. Агилните процеси промовираат софтверски развој кој може да се одржува. Клиентите, девелоперите и корисниците треба да одржуваат константно темпо на неопределено време;
9. Постојаното внимание на техничката точност и добриот дизајн придонесуваат за подобрување на агилноста;
10. Едноставноста, односно уметноста за максимизирање на работата која не е завршена е од суштинско значење;
11. Најдобрите архитектури, барања и дизајни произлегуваат од самоорганизираните тимови;
12. Во редовни интервали размислува како да се биде по ефикасен и тоа однесување го приспособува соодветно.

2.2.2 Видови на агилни методи

Од принципите наведени во Манифестот произлегува и самото дефинирање на поимот за агилното програмирање како и основите на агилните методологии. Методологиите кои ја промовираат агилноста во основа ги имаат

истите принципи, а се разликуваат меѓусебно во нивната практична примена. Во листата на агилните методологии припаѓаат:

- Екстремно програмирање (Extreme Programming XP);
- Scrum;
- Evolutionary Project Management (Evo);
- Unified Process (UP);
- Crystal;
- Lean Development (LD);
- Adaptive Software Development (ASD);
- Dynamic System Development Method (DSDM);
- Feature Driven Development (FDD).

2.2.2.1 Екстремно програмирање

Екстремното програмирање е една од најпознатите агилни методологии. Она што е уникатно за овој метод во однос на останатите агилни методи е тоа што тој се фокусира на софтверскиот развој и не се истакнува како проектна методологија. Всушност станува збор за дисциплина, пристап за развој на софтвер со цел за континуирана и висококвалитетна испорака на софтвер.

Овој пристап се покажал како успешна практика во разни компании и индустрии од најразличен домен. Првиот проект поврзан со екстремното програмирање датира од 6-ти март 1996 година. Станува збор за проектот Chrysler Comprehensive Compensation System (познат уште како „C3“) во американската корпорацијата Chrysler[6] за производство на автомобили и нивни делови. Овој систем бил изграден со користење на објектно ориентираниот програмски јазик Smalltalk и објектно ориентираната база на податоци GemStone, а новите техники кои се користени за софтверскиот развој се од особено значење за софтверското инженерство.

Екстремното програмирање во неговата основна форма се темели на пет основни вредности и тоа:

[6] Од јуни, 2009 г. позната е како Chrysler Group LLC

1. *Едноставност* – Ова е една од суштинските вредности на која се темели екстремното програмирање. Испорачаниот софтверски продукт треба да е едноставен и да ги задоволува наведените кориснички барања. Во центарот на вниманието се ставени тековните кориснички барања и тие се предмет за дизајн и имплементација на тимот, додека сите понатамошни предвидувања и планирања за иднините кориснички барања не се разгледуваат.

2. *Комуникација* – Комуникацијата е исто така значајна особина на екстремното програмирање. Програмерите се во постојана комуникација со клиентите како и со останатите членови од тимот за развој на софтвер.

3. *Повратен одговор* – Развојот на софтверскиот продукт се реализира во итерации и на крајот од секоја итерација тимот за развој на софтвер добива повратен одговор од клиентите. Од овој повратен одговор се започнува наредната итерација.

4. *Храброст* – Една од практиките која ја поттикнува оваа вредност е обврската на дизајнерите и девелоперите да ги извршуваат нивните задачи денес, а не утре. Ова мото воедно означува и поттик за девелоперите да прават промени во кодот за негово подобрување, а воедно и за полесна имплементација на идните промени во системот.

5. *Почит* – Оваа вредност означува себепочитување и почит кон останатите членови од тимот. Исто така, сите членови од тимот за развој на софтвер треба да имаат почит кон нивната работа, а оваа почит значи нивен константен напор за постигнување висок квалитет во работата како и постојано подобрување на имплементираните софтверски решенија.

Успешниот развој на софтвер е дело на тимска работа. Ова значи дека, освен тимот за развој на софтвер, во процесот се вклучени и клиентите и менаџерите. Екстремното програмирање е едноставен процес што ги поврзува овие интересентни групи заедно за постигнување на заеднички успех. Еден XP тим брои од неколку до десетина членови и тој претежно се фокусира на објектно ориентирани проекти.

Клиентите, менаџерите и програмерите работат заедно за успешно имплементирање на бараниот систем [2]. Улогите кои се појавуваат во Екстремното програмирање се следните:

1. *Менаџер*: управува со тимот за развој на софтвер, организира состаноци, се грижи за проблемите со кои се соочува тимот итн.

2. *Тренер*: тренерот е програмер кој ги подучува и следи членовите од тимот во нивното спроведување на XP процесот. Тој е одговорен за техничкото извршување на проектот и треба да има одлични комуникациски вештини. Оваа улога заедно со улогата на трагачот (tracker) го сочинуваат менаџерскиот дел од еден XP проект.

3. *Трагач*: Ова лице е програмер кој ги собира корисничките приказни и тестовите на прифатливост од девелоперите.

4. *Тестер*: Ова лице може, но не мора да е програмер кој им помага на клиентите во креирањето и развојот на тестови за тестирање на софтверот, особено за тестовите кои не можат да бидат автоматски.

5. *Програмер*: Програмерот е лице кое пишува тестови, дизајнира и кодира. Ова лице може да биде и тестер.

6. *Клиент*: Клиентот е задолжен за креирање на корисничките приказни (user stories) и тестови за прифатливост. Клиентот е тој кој им дава приоритет на корисничките приказни и ги распоредува во итерациите на софтверскиот процес.

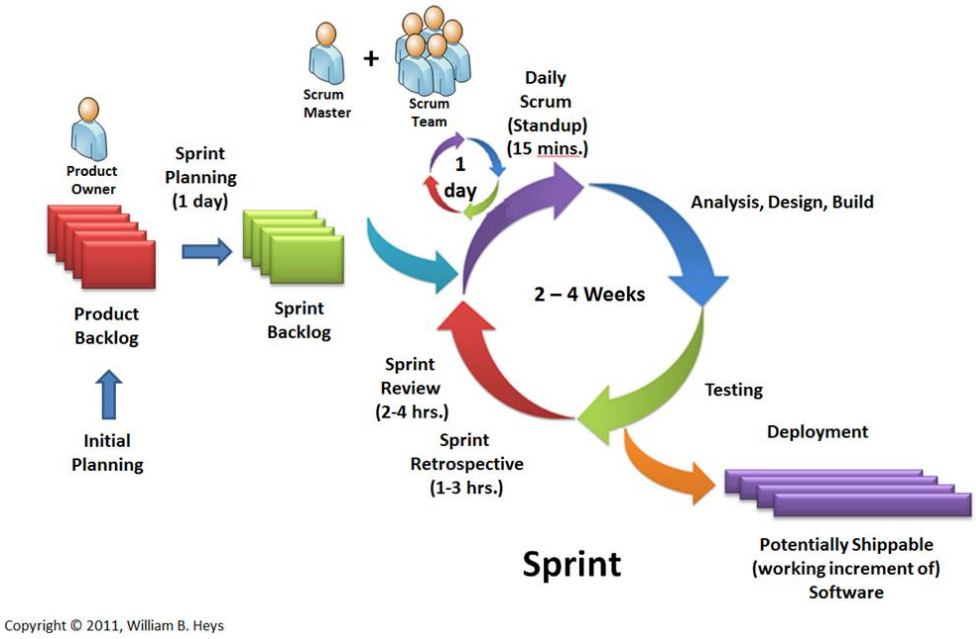
Слика бр. 2 – Животен циклус на еден XP проект

2.2.2.2 Scrum

Почетоците на оваа методологија датираат од 1986 г. кога Takeuchi и Nonaka, кои вовеле нов пристап во развојот на софтвер во кој сите фази на процесот се преклопуваат и тимот заедно работи низ различните фази. Терминот Scrum потекнува од рагбито и означува враќање на топката назад во играта со помош на тимот. Она што денес се смета за Scrum методологија потекнува од почетоците на истражувањата на Ken Schwaber и Jeff Sutherland од 90-тите. Развојниот процес на еден софтверски производ се реализира во три фази и тоа:

1. Пред играта (планирање, дизајн/архитектура големо ниво на апстракција);
2. За време на играта (развој, имплементација, спринтови – итеративни циклуси, подобрувања, нови верзии) и
3. После играта (систем подготвен за продукција).

Развојот на еден проект со користење на Scrum се реализира во итерации каде секоја итерација (sprint) трае од 2 - 4 недели. Ова времетраење мора да биде однапред одредено и фиксно. За тоа време се изработуваат верзии на производот подготвен за испорака, додека целокупниот проект (една игра) трае од 3-8 спринтови.



Слика бр. 3 – Фази во развојниот процес со користење на Scrum

Развојниот процес со користење на Scrum се реализира во неколку фази (Слика бр.3). Почетната фаза се однесува на планирањето на наредната итерација (спринт или уште наречен scrum „ceremony“). Ова планирање се реализира преку состанок со развојниот тим, менаџерите и со клиентот нарачател на софтверскиот продукт. Клиентот е тој кој што ја креира и приоритизира производствената залиха (поточно, листата на кориснички барања). Откако тој ќе ги постави главните својства кои ќе бидат вклучени во наредниот спринт (во времетраење од 2 - 4 недели), scrum тимот се фокусира на зададената цел на спринтот. Самиот тим, покрај тоа што има јасна претстава за задачите кои треба да се извршат за постигнување на целта, тој има и поголема слика за целокупноста на проектот. Откако за време на состанокот ќе се одредат својствата кои ќе бидат цел на тимот за наредните 2 - 4 недели, не е дозволено да се прави повторна приоритизација на својствата сè додека не се заврши спринтот, односно сè додека не заврши триесетдневниот процес на дизајн, имплементација и тестирање на својствата.

Во рамките на еден Scrum тим се генерираат неколку видови на документи во кои припаѓаат:

- Залиха на производите (product backlog) – Еволуирачки, приоритизиран редослед на функционалностите (кориснички барања) кои треба да се

имплементираат во системот, грешките и недостатоците кои треба да се променуваат во текот на софтверскиот развој [6]. За секое барање во залихата на производи постои единствен идентификатор за барање, категорија (својство, подобрување, недостаток (грешка)), статус, приоритет, се проценува потребното време за имплементација. Залихата е организирана во облик на табла.

- Залиха на спринтови (залиха на тековната итерација) – Оваа залиха е листа која се состои од сите работни и технолошки својства, подобрувања и недостатоците кои се распоредени во тековната итерација (sprint). Барањата се зададени во форма на задачи за кои има краток опис, историја на настанувањето, лицето задолжено за нејзино извршување, статус на задачата и времето потребно за нејзино извршување. Оваа залиха се променува секојдневно.
- Дијаграм на преостанатата работа во Спринтот – Го прикажува преостанатото време за завршување на спринт задачите.

Еден Scrum тим работи најчесто заедно, но постојат и примери кога тимот е дистрибуиран на различни локации, па состаноците се спроведуваат преку конференциска врска. Улогите[7] кои се дефинирани во рамките на еден Scrum процес се следните:

- Сопственик на продуктот (product owner) – Станува збор за лице кое е одговорно за креирање и приоритизирање на залихите, односно ова е лицето кое одлучува што ќе се изработи и по кој редослед. Ова лице е задолжено и за избор на сè она што ќе биде вклучено во наредната итерација, односно во наредниот спринт. Откако ќе заврши еден спринт ова лице треба да направи преглед на системот.
- Scrum мастер (scrum master) – Ова лице има за цел да го води тимот за успешно спроведување на избраниот процес, ги отстранува бариерите и проблемите кои се јавуваат во развојниот процес, ги одржува дневните состаноци (scrum meetings), со еден збор, тој треба да претставува олеснувачки фактор во софтверскиот развоен процес. Ова лице не е само

[7] Поделбата на улогите е превземена од податоци наведени на официјалната веб страна на непрофитната организација Scrum Alliance: www.scrumalliance.org

менаџер, туку е и програмер кој учествува во имплементацијата на продуктот.

- Тим (scrum team) – Бројот на членови во рамките на еден scrum тим се движи од 5 до 9 членови. Најчесто еден ваков тим брои 7 членови. Овој тим има за цел постигнување на целта на самиот спринт со што има одврзани раце за постигнување на истата цел во рамките на ограничувањата на системот.

Слика бр. 4 – Scrum улоги

Оваа методологија е едноставна, лесно разбирлива и е фокусирана на најзначајните функционалности кои развојниот тим ги воочува и имплементира. Една од значајните карактеристики е, секако, комуникацијата која помага во решавање на проблемите кои се јавуваат во развојниот процес во текот на реализирање на состаноците.

2.2.2.3 Crystal

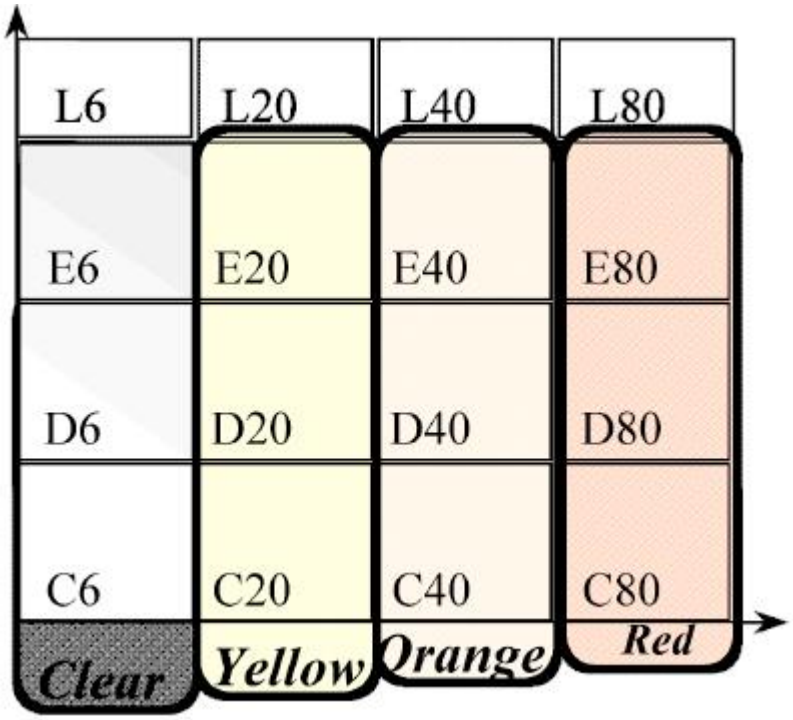
Во 90-тите години на минатиот век, консалтинг групата на IBM го вработила Alistair Cockburn за да направи посебна методологија за објектно ориентиран развој на софтвер. Тој направил истражувања на повеќе софтверски проекти и им дал за задача на тимовите да ги посочат причините за успехот на тие проекти. Така, Cockburn ја дефинирал фамилијата на методологии позната како „Crystal“ [5], каде секоја методологија е специфична и се користи за различни типови на проекти. Заедничко за сите овие методологии е тоа што тие

го потенцираат учеството на корисниците во софтверскиот развој и се фокусира претежно на луѓето, интеракцијата, вештините, комуникацијата итн. Исто така, за сите нив важи паролата: „*strong on communication, light on work products".* За разлика од XP методологијата каде имаме строго дефиниран редослед на развојни правила, овде имаме поголема слобода на развојниот тим [4]. Оваа методологија лесно се усвојува кај еден тим за развој на софтвер, додека XP методологијата гарантира подобри резултати и зголемена продуктивност.

Овие Crystal методологии биле развиени за да се применат на специфични проекти. Според Cockburn, не може да постои „една-големина-одговара-на-сè" развоен процес. На сликата[8] број 5 ни е прикажана фамилијата на Crystal методологии [5]. Името „Crystal" е добиено од двете димензии прикажани на сликата подолу, како аналогија на геолошките кристали кои се разликуваат по две особини: боја и цврстина. Големината на тимот (поточно комуникациската и координациската сложеност) соодветствуваат на бојата, а критичноста соодветствува на цврстината на минералите: системи од типот „lost-of-comfort" соодветствуваат на мекиот кварц, додека системите од типот „loss-of-life" на цврстите дијаманти. Графиконот на сликата помага во изборот на соодветниот „Кристален" метод. На апцисата ни е даден бројот на членовите во тимот за развој на софтвер, а на ординатата ни е претставена критичноста, односно потенцијалната штета (губиток) која би произлегла од системот. Како што се придвижуваме на десно од x-оската така и се зголемува бројот на членови од тимот за развој, а ова значи зголемено документирање на развојниот процес, употреба на разни алатки за координација и намалена меѓусебна комуникација. Поаѓајќи од најниските до највисоките вредности на y-оската ние ги идентификуваме губитоците, односно најмала форма на губиток би предизвикала недостаток на комфорност, потоа средна форма би предизвикала загуба на дискрециони финансиски средства, потоа загуба на големи финансиски средства, а најголемата е „загуба на живот". Ознаките C (lost of comfort), D (loss of discretionary money), E (loss of essential money) и L (lost of life) се однесуваат на овие форми на критичности. Со зголемување на степенот на критичноста сè до L вредностите, се променуваат верификационите и

[8] Превземено од книгата на Cockburn, A. (2004). Crystal "clear": A human-powered software development methodology for small teams. Addison-Wesley (стр.240)

валидациските компоненти. Па, така, изборот на дадена методологија од Crystal фамилијата зависи од бројот на членови во тимот и критичноста. На сликата подолу ни се претставени вкупно 16 методологии (4x4 квадрати означуваат 16 методологии). Најпознатите методологии се: „Кристално чиста“ (Crystal Clear), „Кристално портокаловата“ (Crystal Orange) и „Кристално црвената“ (Crystal Red).



Слика бр. 5 – Crystal фамилијата на методи за различни типови на проекти

2.2.2.4 Adaptive Software Development (ASD)

Адаптивниот развој на софтвер [9,10] бил воведен од страна на James Highsmith in 1997г. и се базира на брз апликациски развој (Rapid Application Development). Оваа методологија го поддржува инкременталниот и итеративен развој користејќи го притоа прототипирањето. Софтверскиот развој со оваа методологија се реализира во три фази. Поимите со кои се искажани овие фази алудираат кон самите улоги во проект воден од една ваква методологија.

1. *Истражување и размислување* – истражување и размислување наместо долгорочно планирање;
2. *Соработка* – координирање и комуницирање со тимот и
3. *Учење* – континуирано учење и подобрување.

Слика бр. 6 – Животен циклус на ASD методологијата

Фаза на истражување и размислување

Во првата фаза од адаптивниот софтверски развој постојат пет основни чекори. Секој чекор од фазата на размислување и истражување може да се промени повеќе пати за време на започнувањето со проектот и планирањето. Во почетокот на започнување со проектот треба да ги поставиме целите на проектот, да го поставиме текот на организирањето, да ги претставиме ограничувањата, специфицирање и дизајн на барањата итн. Во зависност од големината на проект, времето за започнување на еден проект варира од 2-5 дена за мали и средни проекти и 2-3 недели за поголеми проекти. Потоа треба да го дефинираме бројот на итерации и временскиот рок за секоја од овие итерации. Големината на целокупниот проект и стапката на несигурност се два фактори кои ја одредуваат должината на секоја итерација.

Откако ќе се утврди бројот, распоредот и должината на секоја итерација, тимот за развој и нарачателот ги одредуваат целите на итерациите, односно нарачателот на софтвер ги разгледува сите функционалности кои произлегуваат како резултат на итерациите. На секоја итерација развојните програмери доделуваат задачи кои ќе треба да се имплементираат чија приоритизација е одредена од страна на нарачателот на софтвер.

Слика бр. 7 – Фази во животниот циклус на методологијата за адаптивен софтверски развој

Фаза на соработка

Раководителот на тимот помага во соработката како во развојниот процес, така и во односот со интересентните групи. Соработката со помали тимови се реализира преку неформални разговори, додека кога станува збор за поголеми тимови потребни се дополнителни техники и алатки за интеракција. Развојот на софтвер не подразбира само вклучување на програмерите во развојот, туку овде имаме интеракција и вклучување на сите интересентни групи: нарачател на софтвер, екстерни консултанти, менаџери итн.

Фаза на учење

Кога станува збор за водопадниот модел имаме минималност во однос на грешките кои настануваат во развојниот процес бидејќи овде имаме еден линеарен софтверски развој. Меѓутоа, не постои програмски код (особено за вакви големи системи) кој нема багови или грешки, па, според овој модел, се настојува сè да биде добро направено уште од првиот обид. Така, постоењето на грешки не значи неуспех на развојниот тим, туку можност за натамошно подобрување и учење. Развојниот тим учи од своите грешки, ги подобрува и редуцира истите во иднина. Неригорозноста во однос на постоењето на грешки во програмскиот код остава простор за понатамошно усовршување и подобрување на развојниот тим.

На крајот од секоја итерација можат да се извлечат разни информации за: квалитетот на резултатот од корисничка и техничка перспектива, потоа за функционирањето на тимот за развој и за статусот на проектот. Главен приоритет на методологијата за адаптивен развој на софтвер е повратната информација која ја дава корисникот преку неговата директна интеракција со софтверскиот продукт. Токму оваа повратна информации дава оцени за квалитетот на продуктот и насоки за натамошниот развој, што е подобра опција од документацијата и дијаграмите. За да се оцени квалитетот на софтверскиот продукт од технички аспект потребни се проверки на програмскиот код, потоа на целокупниот дизајн и архитектура на системот. Функционирањето на развојниот тим се разгледува од аспект на квалитетот на софтверскиот развој за кој тимот е задолжен. На крајот од секоја итерација тимот треба да направи преглед на насоката по која се реализирал програмскиот развој, односно да согледа и да предложи начини за понатамошно подобрување како на програмско, техничко ниво така и на социјално, тимско ниво. Статус на еден проект значи да се види до каде стигнал програмскиот развој, дали тој е во согласност со корисничките барања, дали се реализира по саканиот план за работа, каде треба да биде проектот итн.

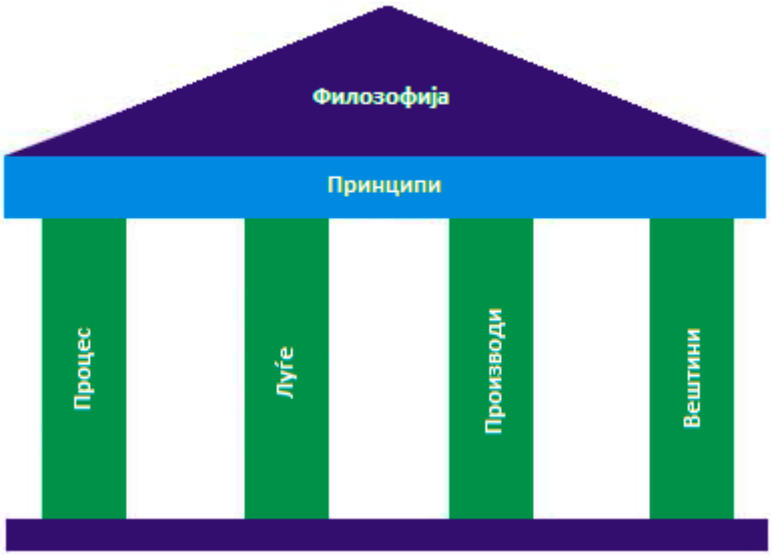
2.2.2.5 Метод на динамички развој на систем (DSDM)

Методот на динамички развој настанал како одговор на RAD (Rapid Application Development) заедницата во 1993г. Оваа методологија била објавена во 1995 година и е позната по тоа што била агилна методологија која настанала пред Агилниот манифест и била создадена од страна на членовите на непрофитната организација DSDM конзорциум [12].

Методот на динамички развој на систем (DSDM) е агилна методологија која има пет основни чекори на кој ѝ претходи фазата на пред-проект и ја следи фазата пост-проект [13]. Петте проектни чекори се: Студија на изводливост, Бизнис план, Функционален итеративен модел, Дизајн и Развојна итерација и Имплементација. Во развојната рамка на активности, активностите се поддржани од деветте принципи во кои припаѓаат:

1. Активно учество на корисникот;

2. Мотивиран тим кој донесува одлуки;
3. Честа испорака на производи;
4. Испорака на производ во согласност на корисничките барања;
5. Итеративен и инкрементален развој;
6. Сите промени за време на развојот можат да се поништат (Реверзибилни промени);
7. На почетокот на проектот се дефинираат основните барања;
8. Тестирањето и интеграцијата се спроведуваат во текот на целиот животен циклус на еден проект и
9. Соработката и комуникацијата помеѓу сите интересентни групи е од големо значење.



Слика бр. 8 – Развојна рамка на DSDM методологијата

Фази во животниот циклус на DSDM методологијата

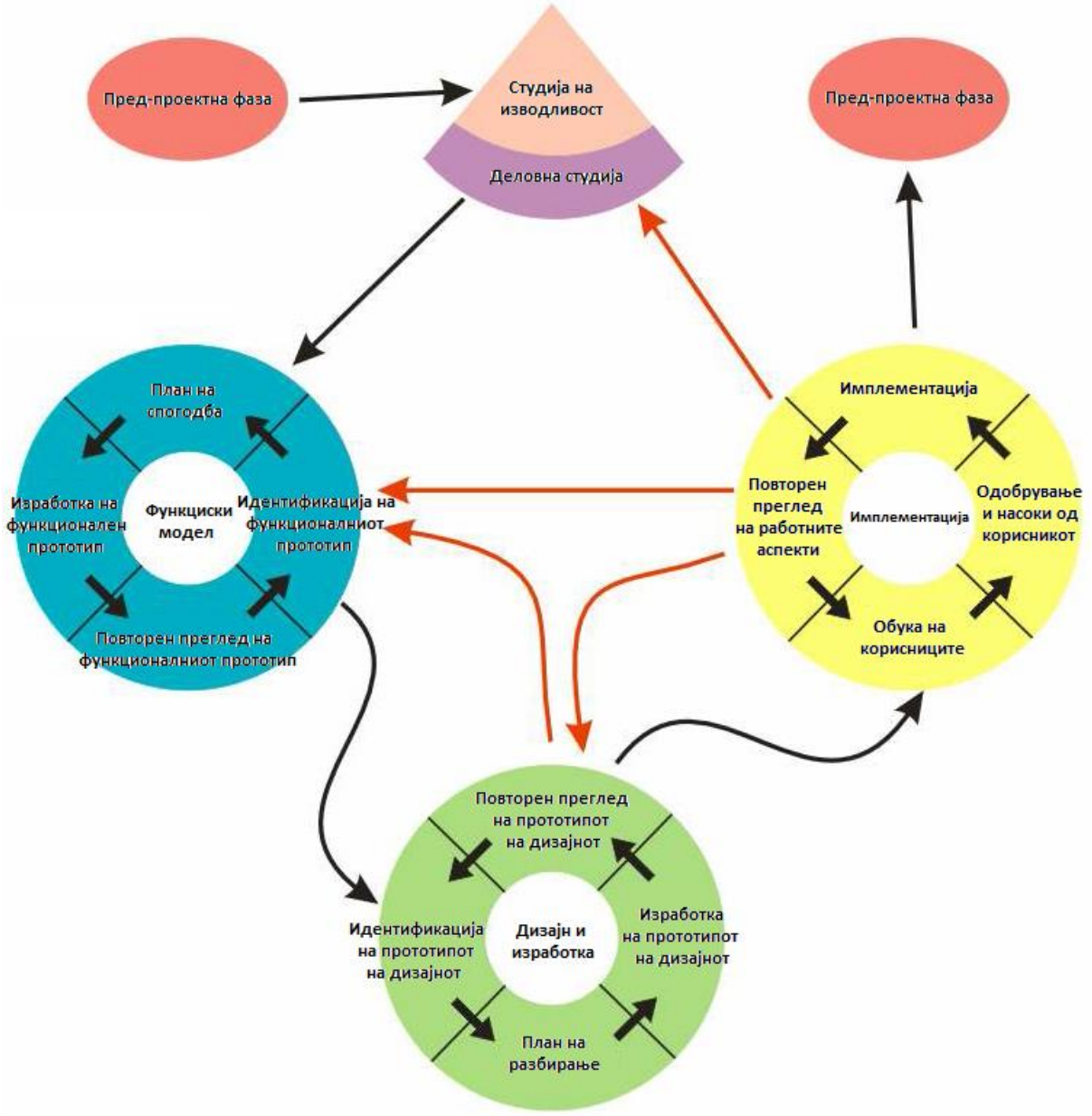
DSDM методологијата се фокусира на ИТ проекти со кратки рокови и ограничен буџет. Во рамките на еден проект имаме три основни секвенцијални фази:

1. Пред-проектна фаза;
2. Проектна фаза:
   2.1. Студија на изводливост,
   2.2. Деловна студија,
   2.3. Итерирање на функционалниот модел,

2.4. Итерирање на дизајнот и развојот,

2.5. Имплементација и

3. Пост-проектна фаза.



Слика бр. 9 – Животен циклус на еден DSDM проект

Во првата пред-проектна фаза се дефинира буџетот и задолженијата за негова реализација.

Проектната фаза се состои од пет пофази. Првите две подфази се комплементарни една на друга и се однесуваат на исплатливоста на изработка на еден проект во однос на можните ризици и на истражувањето на целниот пазар соодветно. Третата подфаза се однесува на идентификација на

функционалностите, изработка и проверка на функционалниот прототип. Исто така се одредува и временскиот распоред за развојот на поединечните функционалности. Четвртата подфаза се однесува на идентификација, изработка и проверка на дизајнскиот прототип. Последната подфаза се однесува на имплементацијата, односно се проверува дали изработениот проект е во согласност со поставените цели и кориснички барања.

Последната пост-проектна фаза овозможува ефикасно и ефективно функционирање на системот што се остварува преку одржување, подобрување и поправки.

2.2.2.6 Feature Driven Development (FDD)

Развојот базиран на својства (Feature Driven Development - FDD) е *клиентски* и архитектски ориентиран, реалистичен софтверски процес. Она што беше поимот *корисник* во Екстремното програмирање, овде е интересентна група, а се однесува на поимот *клиентски* од наведената дефиниција. Оваа методологија за првпат била претставена во 1999г. во книгата Java Modeling In Color with UML [8]. FDD методологијата понатаму е развиена од страна на Stephen Palmer и Mac Felsing (2002 г.).

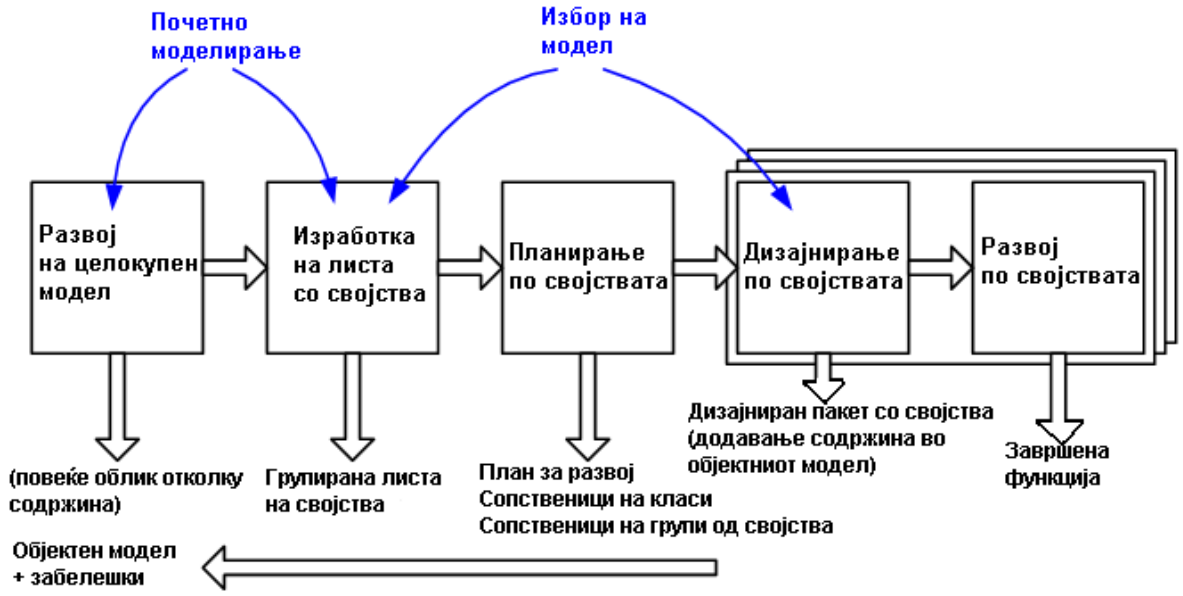
Како што кажува и самото име на оваа методологија, *својствата, карактеристиките (features)* се значаен аспект од оваа методологија. Својството претставува мала, клиентско-значајна функција која може да се имплементира во рок од 2 недели или помалку. Именувањето на едно својство се прави со користење на следниот темплејт:

<action> the <result> <by|for|of|to> a(n) <object>

Во овој израз терминот објект се однесува на лице, место или предмет (на пр. улоги, момент од некој временски интервал итн.). Примери на вакви својства се: Пресметка на вкупната вредност од продажба, валидација на внесената корисничка лозинка, авторизација на трансакции на корисник итн.

Она што е својство за FDD методологијата е истото она што е кориснички случај на употреба (use case) на RUP[9] (Rational Unified Process) и кориснички приказни (user stories) во Екстремното програмирање.

На сликата подолу ни се прикажани 5-те главни инкрементални и итеративни активности на FDD процесот [7,8] .



Слика бр. 10 – Животен циклус на еден FDD проект

(Copyright 2002-2005 Scott W. Amber, Original Copyright S. R. Palmer & J.M. Felsing)

1. Активност бр. 1: Оваа иницијална активност се однесува на *Развојот на целокупен модел.* Како што започнуваме со развојот на проектот, најпрвин треба да го разбереме доменот во чии рамки ќе се имплементира проектот. За таа цел тимот за развој работи заедно со стручни лица од специфичниот домен. Тимот за развој на софтвер се дели на помали групи, вообичаено по 3 лица во група. Целта на поделбата е креирање решенија за секое подрачје за проектот од специфицираниот домен. По пат на консензус се одбира најдоброто решение. Така по редослед сите подрачја од доменот се стават на тема, се избира најдоброто понудено решение и на крај сите овие решенија се обединуваат во еден целокупен сеопфатен модел.

[9] Методологија за софтверски развој базирана на UML (User Modeling Language) која е итеративна, насочена кон архитектурата на апликацијата и базирана на кориснички случаи на употреба (use cases).

2. Активност бр. 2: Наредната активност се однесува на *Изработката на листа со својства (карактеристики).* Откако сме ја поминале првата активност, следно што треба да направиме е да ги идентификуваме сите својства, карактеристики, односно, поточно кажано, сите „кориснички барања“ кои ќе треба да ги имплементираме во еден сублимат т.н. „програмски“ производ. Целиот систем се разделува на помали подсистеми кои содржат дадена функционалност која треба да се имплементира. Временскиот интервал предвиден за дадена функционалност е 2 часа до 2 недели.
3. Активност бр. 3: Трета активност гласи: *Планирање по својствата.* Ова значи дека менаџерот на проектот, менаџерот на развојот и главниот/те програмер/и, кои се дел од тимот за планирање, ќе направат план за редоследот по кој ќе се развиваат својствата (функционалностите). Тие креираат т.н. клучни точки (milestones) за итерациите од типот „дизајнирај по својство, развивај по својство“.
4. Активност бр. 4: *Дизајнирање по својства* е името на наредната активност од FDD процесот на развој. Во оваа активност се изработуваат секвенцијални дијаграми за секоја функционалност. Откако се дефинира дизајнот програмерите се запознаени со структурата на она што треба да се имплементира.
5. Активност бр. 5: Оваа активност се насочува кон *Развој по својствата.* Откако со претходните активности имаме дефиниран дизајн на системот, започнуваме со имплементација на функционалностите. Придружна активност на оваа активност е, секако, прегледот на програмскиот код (code inspection) и поединечното тестирање (unit testing). Сето ова придонесува за главниот програмер да даде зелено светло за понатамошно насочување на програмскиот код кон build процесот.

Најголем дел од вложениот труд и напор (околу 75%) за еден FDD проект се доделува на 4-тата (*Дизајнирање по својства*) и 5-тата (*Развој по својствата*) активност од претходно опишаниот процес. Погоре опишаните активности честопати се идентификуваат со терминот *процес,* па така при поделбата се наведува дека самата FDD методологија се состои од 5 процеси.

3 Основи на Канбан методологијата

3.1 Lean концепт

Пораките од документот Agile Manifesto произлегуваат од принципите на lean менаџментот во софтверското инженерство. Lean менаџментот најпрвин се појавил во индустријата во Јапонија во 80-тите години на минатиот век од каде се изведени и принципите на овој менаџмент. Lean концептот можеме да го дефинираме како множество од алатки кои придонесуваат за препознавање и елиминација на губитокот. Со отстранувањето на губитокот се зголемува квалитетот, додека времето за производство заедно со трошоците се намалуваат. Такви алатки се, на пример, Value Stream Mapping, Five S, Kanban (pull систем) итн.

Поимот *lean* најпрвин бил употребен во книгата „The machine that changed the world“ на J. P. Womack и D. T. Jones. Оваа книга произлегла како резултат на петгодишната истражувачка студија на IMVP (International Motor Vehicle Program) тимот на MIT[10] Универзитетот. Авторите овде го употребиле терминот *lean production* како синоним за производствениот автомобилски систем на Тојота[11]. Во студијата е направена споредба на западната автомобилска индустрија и онаа на Toyota и визионерот Taichi Ohno.

Во софтверското инженерство поимот *lean* оригинално се среќава во книгата Lean Software Development на Mary и Tom Poppendieck од 2003 година [22]. Принципите на Lean концептот во софтверското инженерство се слични на оние во производството. Такви се, на пример, следните 7 принципи:

1. Отстрани го губитокот,
2. Зголеми го учењето,
3. Одлучувај што е можно најдоцна,
4. Испорачај што е можно побрзо,
5. Поттикни го тимот,
6. Вгради интегритет и
7. Преглед на целината.

[10] Massachusetts Institute of Technology
[11] TPS (Toyota Production System)

Практиките на Lean софтверското инженерство се опишани на сличен начин како и нивните соодветни еквиваленти во агилното софтверско инженерство. Во овие практики се вбројуваат:

- Преглед на губитокот (waste, muda[12]),
- Value Stream Mapping,
- Set-based development,
- Pull systems,
- Queuing theory,
- Motivation и
- Measurements

3.2 Поим за Канбан

Поимот Канбан има јапонско потекло, односно тој доаѓа од јапонскиот збор „Kan“ што значи сигнал и „ban“ што значи картичка или табла. Од овде произлегува дека зборот Канбан е кованица од двата збора кан и бан што заедно во буквален превод означуваат сигнална картичка која се користи како сигнал за тригерирање на акција.

Канбан методологијата првенствено била создадена за производството, но таа своја примена нашла во многу други области. Па, така, денес оваа методологија ја користат архитекти, економисти, менаџери, тимови за развој на софтвер и многу други. Примената од адаптирањето на оваа методологија кај другите науки е многу голема. Во суштина станува збор за методологија која лесно се приспособува на областа во која се применува. Посебен осврт и од особен интерес ќе биде софтверското инженерство каде оваа методологија зазема сè поголем замав. Меѓутоа во софтверското инженерство земајќи ги предвид карактеристиките на еден софтверски продукт не може директно да се мапира Канбан методологијата од сферата на производството во софтверското инженерство. Развојот на софтвер не е продуцирачка ниту, пак, производствена

[12] Еден од трите типови на губиток (muda, mura, mun) кој е клучен концеп во Системот за Производство на Тојота.

активност[13] [19]. Во процесите на креирање на софтверски продукти се прават различни продукти за разлика од производствените процеси каде се произведуваат исти продукти континуирано. Едно од најважните нешта за Канбан методологијата е фактот дека оваа методологија се применува на нашиот веќе постоечки процес. Со оваа методологија не се креира нов начин на управување со постоечкиот процес, туку се пронаоѓаат начини за подобрување на веќе постоечкиот процес. Од овде произлегува фактот дека Канбан не е едноставна како останатите методологии за развој на софтвер или пак некаков пристап во управувањето со софтверските проекти. Бидејќи примената на оваа методологија не подразбира големи промени во тековниот процес каде се применува, тоа всушност посочува на минимален ризик при нејзиното имплементирање, а тоа е од особено значење. Така е на пример, ако се разгледува можноста за примена на Канбан методологијата кај некој тим за развој на софтвер.

3.3 Историски развој

Канбан е концепт кој се поврзува со превземањето на материјали или потребни ресурси само за време (JIT или скратено Just-In-Time)[14] на производството и тоа само за она што има потреба да се произведе, во соодветното време и потребната количина [18]. Системот Канбан е дел од производствениот концепт на JIT. Овој JIT концепт бил започнат од Јапонската фирма Тојота во 1940-тите и тој бил познат како Производствен систем на Тојота (Toyota Production System – TPS). Тој се дефинира како: *филозофија на производството која е базирана на планирана елиминација на сите форми на губиток и на трајно подобрување на продуктивноста* [14]. JIT наоѓа примена во производствени процеси кои се повторливи и се произведуваат исти делови или производи во серија.

Самото потекло на Канбан доаѓа токму од Производствен систем на Тојота. Во позната книга на Taiichi Ohno, Toyota Production System [17], тој ги посочува JIT концептот и автоматизацијата со човеков допир како темели на

[13] Првично превземено од статијата на Jack W. Reeves, "What is Software Design?" C++ Journal, 1992.

[14] Термин уште познат и како линеарно производство (*lean production)* или производство без залиха (*stockless production) .*

Производствениот систем на Тојота (TPS), а Канбан го дефинира како алатката за управување со истоимениот систем:

„The two pillars of the Toyota production system are just-in-time and automation with a human touch, or autonomation. The tool used to operate the system is kanban“ [17].

За да се разбере примената на Канбан во софтверското инженерство, потребно е да се разгледа првичната, оригинална употреба на Канбан во рамките на Производствениот систем на Тојота (TPS).

3.3.1 Канбан во TPS

Во 40-тите години на минатиот век јапонската автомобилска компанија Тојота ја согледала потребата од намалување на трошоците со воведување на техники за правилно складирање на ресурси (proper inventory stocking). Најпрвин тие ги проучувале принципите на работа во супермаркетите и тоа како тие успеваат да имаат полни рафтови со производи и доволна количина од истите за да ги задоволат корисничките конзумирачки потреби. Со понатамошна анализа се дошло до заклучок дека потрошувачите се уверени дека имаат една константна залиха од бараните производи и дека тие само го избираат потребниот број на такви производи кои моментално им требаат. Токму самата корисничка увереност во константната залиха на потребните производи и слободната достапност на производите е она што било искористено како директива во Производствениот систем на Тојота. Имено, компанијата Тојота дошла до заклучок дека ако и нивниот аналоген „супермаркет“ од производствени материјали би можел да даде гаранција за достапност на материјали за нивните производствени линии, тогаш би се намалила потребата за големи складишта и магацини со производствени материјали. Ова автоматски повлекува и намалување на трошоците и побарувањата за складирање.

Токму затоа, Канбан системот се нарекува уште и „Методот на супермаркетот“ бидејќи како што е опишано погоре, самата идеја доаѓа од начинот на работа на супермаркетите. Ваквите продавници со масовна малопродажба користат картички за контрола на производите во кои ги внесуваат информациите за името, кодот, количеството итн. на производите. Бидејќи Тојота користела Канбан сигнални картички за производствените

процеси, самиот метод бил наречен „канбан систем“. Во производствената околина на Тојота, кога еден процес ќе се „обрати“ до претходниот процес за повлекување на потребните делови, тој ја користи Канбан (сигналната картичка) за да дознае кои производи се употребиле.

Промоторот на идејата за JIT, Taiichi Ohno, го вовел овој концепт за изедначување на супермаркетот и потрошувачот со претходниот и наредниот процес соодветно. Со тоа што наредниот процес (потрошувачот) „оди“ кон претходниот процес (супермаркетот) за превземање на потребните делови во време кога се потребни и во бараната количина, тогаш можно е подобрување на ефикасноста на постоечкиот производствен систем. Така претходните процеси нема да нудат вишок на делови и испорака кон наредниот процес.

На сликата подолу ни е претставен еден концептуален дијаграм [18] на канбан систем. Овде имаме два вида на визуелни физички сигнали, односно два вида на Канбан (Продукциско инструкциски Канбан и Канбан за превземање на делови) кои се користат за управување со деловите во производствениот процес.

Слика бр. 11 – Концептуален дијаграм на Канбан системот

3.4 Поим за Push и Pull системи

Push системите се едни од најпознатите системи во производството уште познати и ако системи кои се водени од аспект на произведувачот. Ова значи дека имаме производство на производи на база на предвидено користење од страна на потрошувачите и дел од нив се пласираат на пазарот, а остатокот се складира во разни магацини, складишта итн. Самиот произведувач одлучува што да произведува и во кои количини за да ги задоволи предвидените барања на потрошувачите. Компаниите кои го користат овој пристап мора да ги предвидат побарувањата на производите од страна на клиентите како и квантитетот на истите. Меѓутоа, овие компании со овие предвидувања и проценки честопати не можат да ги задоволат потребите на корисниците, па, како резултат на тоа, имаме незадоволни потрошувачи или пак полни магацини со производи.

За разлика од овие системи, Pull системите се спротивност на Push системите. Тие во суштина се кориснички ориентирани. Ова значи дека производството на производи се случува само кога имаме побарувања од клиентите. Овој систем има за цел задоволување на корисничките потреби и побарувања, користење на помали складишта, намалување на трошоците и постои константно подобрување на дизајнот на производите за задоволување на променливите кориснички барања.

3.4.1 Разлика меѓу Push и Pull системи

Една од главните разлики меѓу push и pull системите е тоа што Push системите имаа за цел производство за *складирање,* додека кај Pull системите производството е одговор на *нарачки* од клиентите. Кога зборуваме за производство од кој било домен, Push производствените системи работат така што самото производство се „турка“, насочува од една операциона фаза кон наредната, без разлика на тоа дали има потреба за производство на производи или не. Кај Pull системите производствената линија се движи во обратна насока користејќи сигнали или картички како тригери за почнување со производство. Обратната насока овде значи дека самиот процес не започнува од добавувачите, суровините итн., туку од магацините со финални производи. Кога

некој потрошувач ќе нарача некој производ, тоа претставува тригер за самиот производствен процес да се поттикне претходната производствена операција за замена на тој производ. Ова значи дека ќе имаме еден производ помалку и тој треба да се замени. Па, така, самиот процес се движи во обратна насока сè до суровините за изработка на производите кои, откако ќе се искористат за производство на производот настанува тригер кој предизвикува порачка на суровини за натамошно производство од самите добавувачи.

Табела бр.1 – Разлика меѓу Push и Pull системи

Push систем	Pull систем
Традиционален систем	Нетрадиционален систем
Предвидено користење	Прецизност во користењето
Поголеми складишта	Помали складишта
Присуство на губиток (waste)	Намалување на губитокот
Непрегледност на проблемите	Визуелна контрола
Предвидено производство	Прецизност во производство
Слаба комуникација	Подобрена комуникација

3.4.2 Предности на Pull системите наспроти Push системите

Типовите на системи Push и Pull имаат свои предности и недостатоци. Предностите од користењето на Pull системи можеме да ги разгледаме од различни аспекти. Еден од тие аспекти е поимот за складирање на произведените производи. Со Pull системите се намалува просторот за складирање во некоја производствена компанија бидејќи немаме вишок во производството. Ова значи и намалување на цената за магацински простор. Друг аспект се корисниците на тие производи. Со имплементирањето на овој систем се зголемува довербата и задоволството кај корисниците бидејќи самите производи се произведуваат за задоволување на нивните потреби. Бидејќи овде имаме производство на помала количина на производи за разлика од Push системите, така и квалитетот на производите ќе се подобрува. Трет аспект кој треба да се напомене е аспектот на прецизност, односно предвидувањето за Pull и Push системите соодветно. Кога станува збор за Push систем, производителите треба да ги предвидат побарувањата на производите од страна на клиентите како и квантитетот на истите. Меѓутоа, ова се само проценки

кои во целост не можат да ги задоволат корисничките барања како во квалитет, така и во квантитет. Па, така, еден Pull систем се одликува со прецизност и точност во однос на производството, а тоа значи заштеда на време кое во еден Push систем се троши за долгорочно планирање. Покрај заштеда на време, кај еден Pull систем имаме и планира продажба на производи за разлика од Push системите каде имаме можна состојба на полни магацини и складишта со производи кои никогаш нема да се продадат. Покрај овие постојат и други аспекти кои ги истакнуваат предностите од употребата на овој тип на систем. Меѓутоа, за систем од некој друг домен, Pull системот може и да не одговара на производствените можности и начинот на самото производство. Дури во некои случаи се препорачува комбинирање на двата типа на системи за максимизирање на целокупната производствена ефикасност и ефективност.

3.5 Теорија за редови

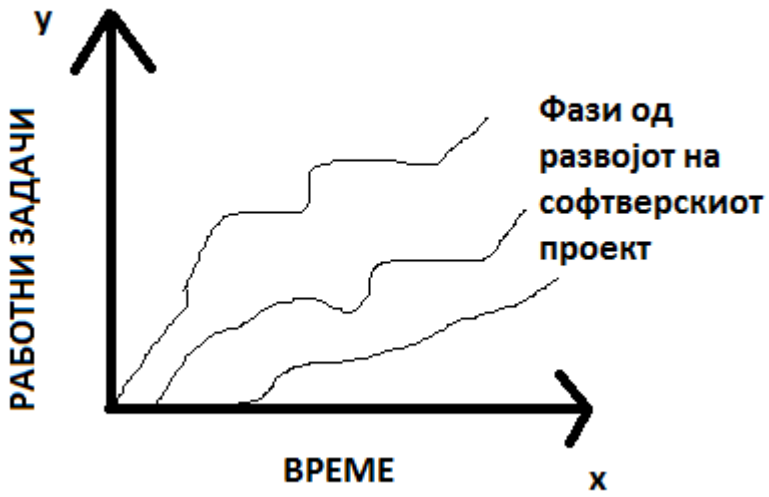
Теоријата за редови (queues) е математичка дисциплина која се занимава со проучување на редовите за чекање (queues). Таа е дел од полето на математиката попознато како менаџмент наука или наука за одлучување (management science, decision science). Примери кои ги вклучуваат овие редови на чекање се:

- Супермаркети – клиенти чекаат за некоја услуга;
- Јавен транспорт – чекање превоз;
- Компјутерски систем – чекање резултат од операција.

Оваа теорија своите зачетоци ги има од истражувањата на данскиот инженер Agner Krarup Erlang со неговите модели за телефонската централа во Копенхаген. Студијата која тој ја напишал во 1990 година е онаа што денес се смета како Теорија за редови.

Во теоријата на редови и во агилните методи за развој на софтвер често користена алатка се *кумулативните дијаграми на тек* (CFD – Cumulative Flow Diagrams). Овие дијаграми се од типот на површински дијаграми (Area charts) кај кои на хоризонталната x-оска го претставуваме *времето,* а на вертикалната y-оска ги претставуваме *работните задачи*. Истите овие дијаграми подетално се

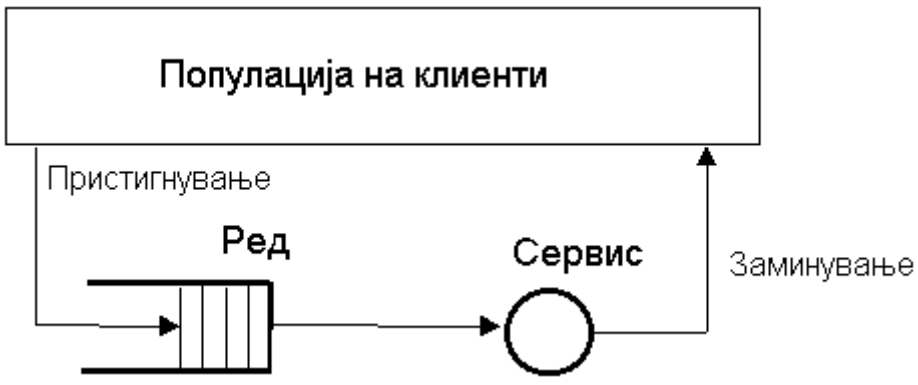
опишани во секцијата Законот на Little's каде се согледува нивната практична примена во рамките на Канбан методологијата.



Слика бр. 12 – Едноставен концепт за еден софтверски кумулативен дијаграм

3.5.1 Поим за систем со ред за чекање (single queuing system)

Во Теоријата на редови, единичен систем со ред за чекање во најопшт случај можеме да го претставиме како модел кој се состои од клиенти кои пристигнуваат во ред за чекање во системот. Откако ќе бидат опслужени (ќе ја добијат потребната услуга, сервис) тие си заминуваат од системот.



Слика бр. 13 – Елементи на единечен систем со ред за чекање

Популација на клиенти – Се однесува на ограничени (затворени системи) или неограничени (отворени системи). Неограничена популација претставува теоретски модел на системи со голем број на потенцијални клиенти (на пример:

банка на прометна улица, бензинска станица на автопат). Пример за ограничена популација може да биде број на процеси кои треба да се извршат (опслужени) од страна на компјутер или одреден број на машини кои треба да бидат поправени од сервисер. Терминот клиенти овде се користи во поширока смисла и може да се однесува на луѓе, машини од различна природа, процеси во еден компјутерски систем, телефонски повици итн.

Пристигнување – Го претставува начинот на кој клиентите влегуваат во системот. Повеќето од овие пристигнувања се случајни со случајни временски интервали помеѓу две последователни пристигнувања. Најчесто овие пристигнувања се опишани преку случајна дистрибуција на интервали наречена шема на пристигнување (Arrival Pattern).

Ред – Претставува одреден број на клиенти кои чекаат за услуга (овој ред може да биде и празен, односно да нема клиенти кои чекаат). Клиентот кој што е опслужен не е дел од редот. Понекогаш клиентите буквално формираат „ред“, како што е пример со редица пред шалтер во банка. Во друг случај клиентите се дел од ред кој што е апстракција (имагинарно пресликување) на „редот“ од секојдневниот живот (на пример: авиони кои чекаат писта за слетување). Постојат две главни одлики на редот: Максимална големина и Дисциплина на редот.

Максимална големина на редот (уште позната како Капацитет на системот) – е максималниот број на клиенти кои можат да чекаат во еден ред (вклучувајќи го/ги и оној/ие кој во моментот се опслужува/ат). Теоретскиот модел на редот претпоставува неограничена должина на еден ред, меѓутоа, во пракса редот е секогаш ограничен. Ако должината на еден ред е ограничена, дел од клиентите ќе бидат откажани без да бидат опслужени.

Дисциплина на ред – претставува начин на организација на редот (правила на влегување и излегување на клиенти до и од редицата). Постојат следниве начини на организација:

1. FIFO (First In First Out), уште наречени FCFS (First Come First Serve) – првиот што ќе дојде прв ќе биде опслужен.

2. LIFO (Last In First Out), уште наречени LCFS (Last Come First Serve) – последниот што ќе дојде прв ќе биде опслужен (стек).
3. SIRO (Serve In Random Order) – опслужувањето се прави по случаен редослед.
4. Priority Queue – приоритетни редици.

3.5.2 Законот на Little (Little's law)

Во математиката, поточно во Теоријата на редови, постои една теорема, закон, формула или уште наречена лема, која прв ја докажал John Little во 1961 година од каде доаѓа и нејзиното име. Овој закон е фундаментален и наоѓа примена во многу области. Тој гласи вака:

Просечниот број на корисници во еден стабилен систем L е еднаков на просечната ефективна[15] рата на пристигнување λ, помножена по просечното време кое корисниците го поминуваат во системот W.

$$L = \lambda * W$$

На пример, ако имаме една продавница во која потрошувачите (луѓето) пристигнуваат со рата од 10 потрошувачи на час и остануваат во продавницата половина час, тогаш од Законот на Little можеме да го најдеме средниот број на потрошувачи во продавницата односно:

$$L = \lambda * W = 10 \frac{\text{потрошувачи}}{\text{час}} * 0.5 \text{ часа} = 5 \text{ потрошувачи}$$

Резултатот од овој закон наоѓа примена во кој било систем или подсистеми во рамките на еден систем. Меѓутоа, системот за кој се применува овој закон мора да биде во стабилна состојба. Ова значи дека системот не треба да е во почетна фаза од развој или пак во крајна фаза на негово затворање. Не треба да постои транзитна состојба и ратата на доаѓање на корисници треба да е иста со рата на заминување на корисници. Ако го земе примерот со

[15] Во примерот со продавницата, постои разлика меѓу рата на пристигнување и *ефективна* рата на пристигнување на потрошувачите. Рата на пристигнување е рата со која корисниците пристигнуваат до продавницата, додека ефективна рата на пристигнување е онаа кога потрошувачите влегуваат во системот. Ако имаме некој систем кој е бесконечен, без одбивање на корисниците (без загуба), тогаш овие два поими можат да се изедначат. Примерот, како и дефиницијата на законот, се превземени од Wikipedia.org.

продавницата, доколку рата на пристигнување на потрошувачите е поголема од онаа на заминување на потрошувачите, тогаш имаме еден нестабилен систем за кој не можеме да го примениме законот и би имале состојба на бесконечно зголемување на бројот на потрошувачи кои ќе чекаат да бидат опслужени.

Во однос на примената на овој закон во еден Канбан процес, овде законот не се употребува во неговата оригинална форма, односно тој се интерпретира на следниот начин:

$$Cycle\ time = \frac{Work - In - Progress}{Throughput}$$

Од овде следува дека циклусното време[16] (Cycle time) е правопропорционално со Работата – Во – Тек (Work –In – Progress), а обратно пропорционално со Пропусноста (Throughput). Со помала Работа – Во – Тек имаме и помало циклусно време, а тоа повлекува и побрз повратен одоговор, поголема визуелност и зголемена Пропусност.

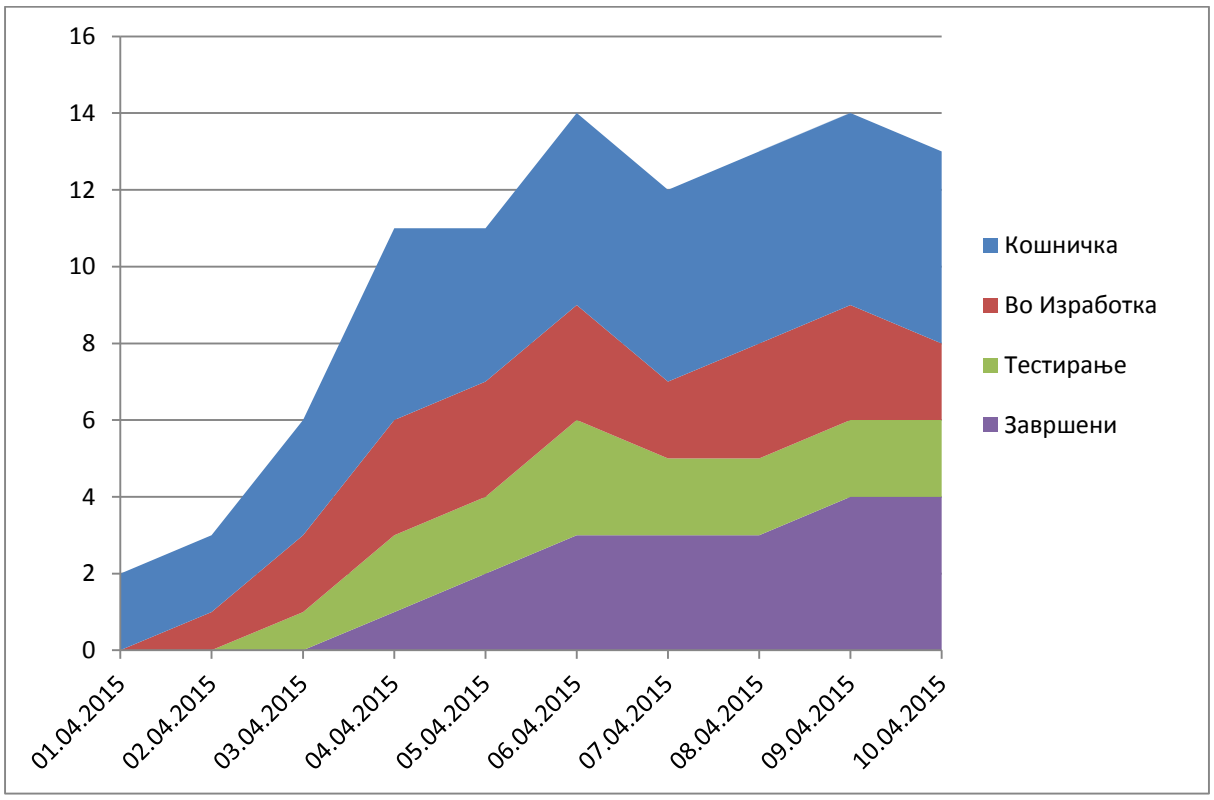
Како што споменавме во подглавјето за Теоријата на редови, кумулативните дијаграми се од особено значење бидејќи даваат увид во тековниот работен статус: *колку е извршено*, *што се извршува тековно* и што уште *чека на извршување*. Овие дијаграми даваат приказ на тесните грла во работниот процес и помагаат во нивното идентификување и подобрување.

На сликата подолу ни е претставен еден пример на кумулативен дијаграм со тест податоци од апликацијата КанбанМАК изработен во Microsoft Excel 2010. На хоризонталната оска ни е претставено времето (конкретно приказ на временска рамка од 10 денови), додека на вертикалната пресликан е бројот на работните задачи. Тест податоците користени при креирањето на дијаграмот се прикажани во табелата што следи.

[16] Поимот за циклусно време е дефиниран подетално во секцијата 2.6.1

Табела бр. 2 – Тест податоци за Кумулативниот дијаграм на слика бр. 14

Дата	Кошничка	Во Изработка	Тестирање	Завршени
01/04/2015	2			
02/04/2015	2	1		
03/04/2015	3	2	1	
04/04/2015	5	3	2	1
05/04/2015	4	3	2	2
06/04/2015	5	3	3	3
07/04/2015	5	2	2	3
08/04/2015	5	3	2	3
09/04/2015	5	3	2	4
10/04/2015	5	2	2	4



Слика бр. 14 – Кумулативен дијаграм

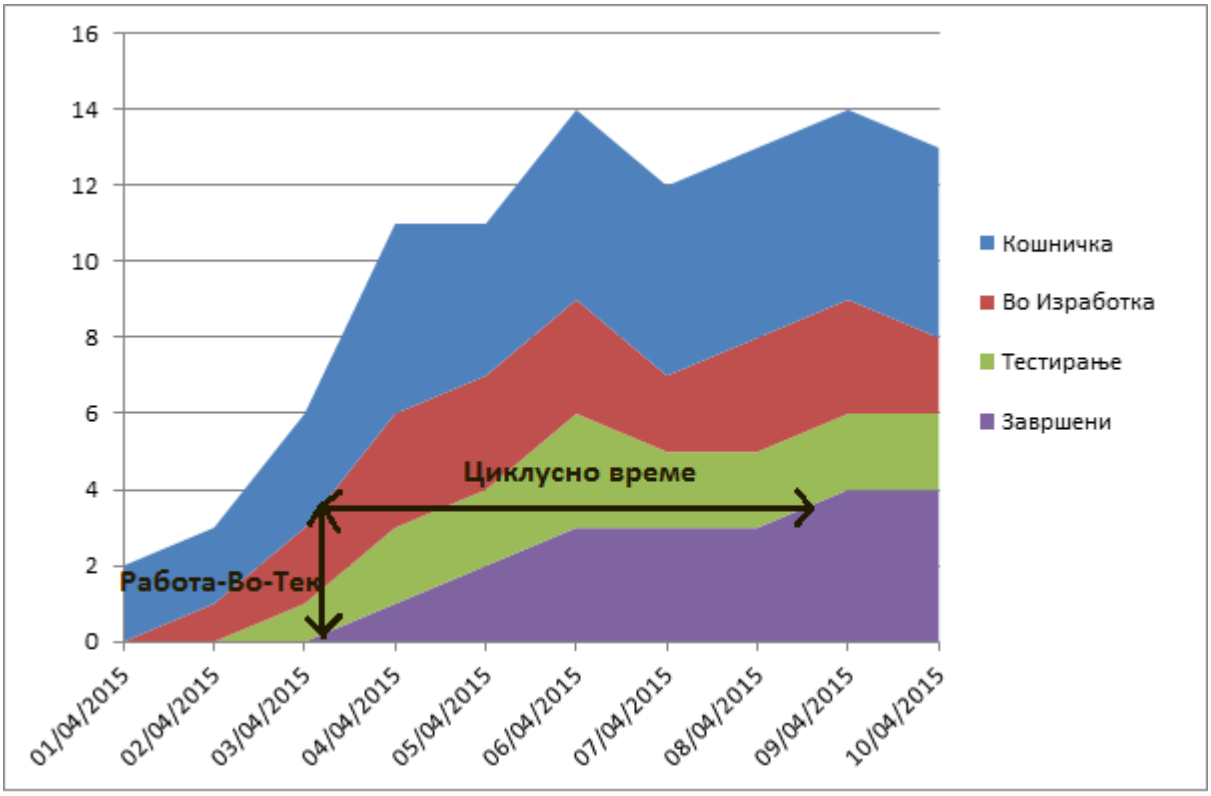
Од дијаграмот прикажан на сликата бр.14 можеме да видиме колку работни задачи имаме во секоја фаза од софтверскиот развој и зголемувањето

на тој број со тек на време. Во КанбанМАК апликацијата имаме 4 колони во Работната табла, па така и овде имаме 4 различни бои на фазите кои ги претставуваат задачите во соодветните колони. Најзначајна фаза, односно боја на област од овој дијаграм е црвената бидејќи таа ги претставува работните задачи кои се во развој. Токму оваа област ни ја претставува *Работата – Во – Тек (Work –In – Progress)* во која се наоѓаат сите оние работни задачи кои се започнати, но не се завршени.

За да ја согледаме визуелно значајноста на оваа „црвена област“, на Слика бр.15 ни е претставен истиот дијаграм од Слика бр. 14, но сега со означување на Работата –Во –Тек и Циклусното време низ самиот дијаграм. Овие два поими се од особено значење бидејќи со нивото зголемување/намалување се согледува самата стагнација/напредок на развојот на проектот. Од законот на Little's кој го дефинираме погоре следеше дека помала Работа – Во – Тек значи и помало Циклусно време што овде би значело следново:

Со одржување на што е можно помали вредности за Работата – Во – Тек и Циклусното време се избегнуваат „тесните грла“ и застои од секаков вид, а тоа придонесува за зголемена пропусност и транспарентност на развојот на проектот.

Исто така од Слика бр.15 може да се забележи и дека „црвената област“ е во *опаѓање и намалување,* што во превод значи помала веројатност од застои и одложувања на проектот. Доколку пак имаме тенденција за зголемување и ширење по вертикала на оваа област тоа значи и поголема веројатност за појавување на „тесни грла“ и застои на развојот на проектот.



Слика бр. 15 – Работа-Во-Тек и Циклусно време за кумулативен дијаграм

3.6 Канбан метрика

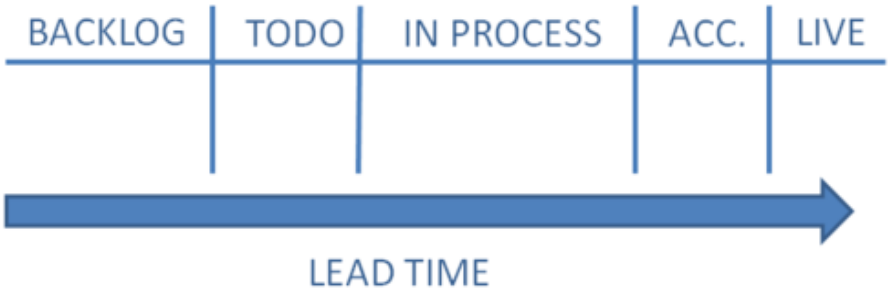
Во еден тим за развој на софтвер, од особено значење е факторот време за испорака на софтвер. За разлика од еден традиционален тим за развој на софтвер кој испорачува дадени функционалности за даден временски период, Канбан тимот се обврзува на испорака на дадена единична функционалност. За ова потребно е да го предвидиме времето од иницијализација на барањето за дадена функционалност од нејзината испорака. Ова е *времето на управување* (*Lead time*). Покрај ова време потребно е да го мериме и времето од започнување со работата за дадена функционалност до нејзиното завршување, односно испорака, а со ова ние всушност го мериме *циклусното време (Cycle time)*. Подетално објаснување за овие времиња следи во наредната секција.

3.6.1 Поим за Lead и Cycle време

Времето на управување (Lead времето) започнува кога едно барање се креира, а завршува со испорака на истото. Ова е времето кое го гледа клиентот

како нарачател на даденото барање, т.е., на пример, некоја функционалност на даден систем.

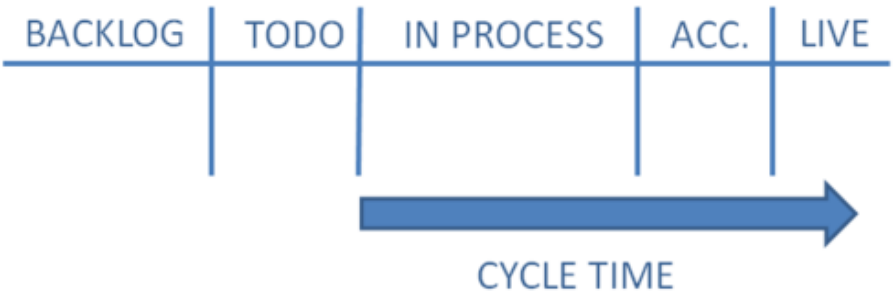
Во еден Канбан систем ова е времето кога барањето[17] е додадено во нашата фиксна Канбан залиха (backlog) и сè до неговиот животен век. Часовникот на ова време започнува со креирањето на барањето, а завршува со испораката на истото.



Слика бр.16 – Време на управување (Lead time)

Циклусното време (Cycle времето) започнува со самото започнување со работа за извршување на даденото барање и завршува кога тоа е подготвено за испорака. Ова време е еден вид мерка за креирање на некоја функционалност или, пак, завршување на некоја задача.

Во еден Канбан систем ова е времето кога барањето се наоѓа во колоната In Process од Канбан таблата и некој започнува со работата на тоа барање. Часовникот на ова време започнува со започнување со работа за барањето и завршува кога тоа барање ќе се креира односно ќе „оживее“.

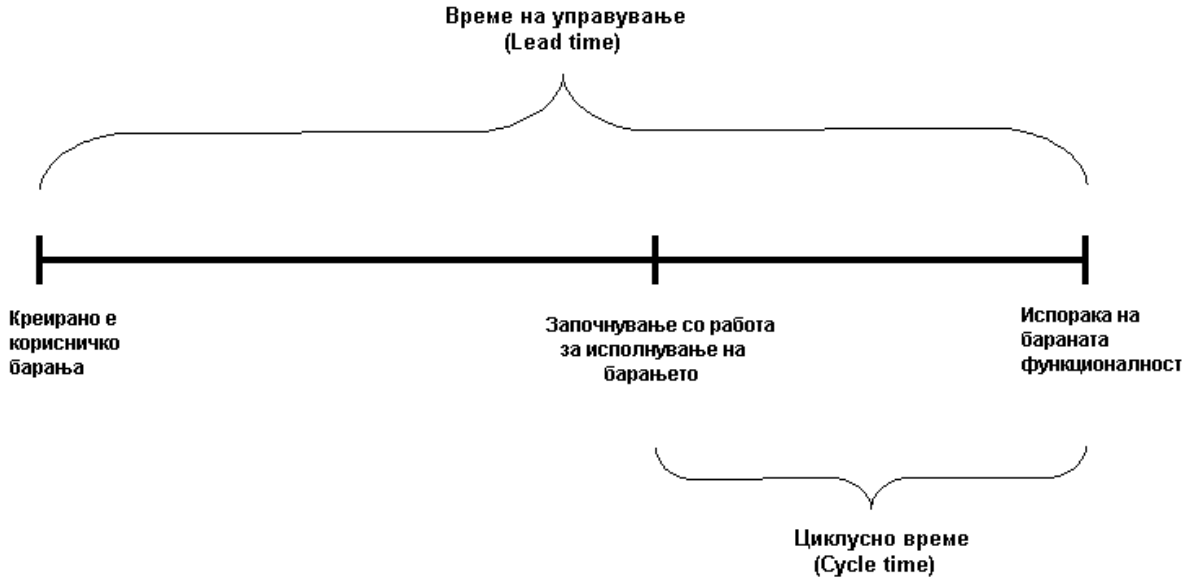


[17] Поимот барање се однесува на поимот задача или работен предмет (work item) кој во англиската терминологија често се употребува како општ поим за означување на дадена функционалност, карактеристика, компонента за дадед софтверски систем.

Слика бр.17 – Циклусно време (Cycle time)

3.6.2 Разлика меѓу Lead и Cycle време

Една од очигледните разлики меѓу овие две времиња се различните единици за изразување на мерењата. Lead времето се мери преку поминатото време (минути, часови, итн.). Cycle времето се мери врз основа на трудот кој го „трошиме“ за креирање на дадено барање дефинирано со време/барање (минути/задача, часови/функционалност). Друга разлика е тоа што времетраењето на времето на водење, управување е секогаш поголемо од циклусното време, односно циклусното време е секогаш помало од времето на управување.



Слика бр. 18 – Време на управување и циклусно време

3.7 Принципи во Канбан методологијата

Во книгата на David Anderson, Kanban - Successful Evolutionary Change for your Technology Business [20], тој дефинира 5 суштински особини кои се појавуваат во сите успешни имплементации на Канбан. Меѓутоа, во истоимената книга на Anderson тој не го користи зборот принцип при дефинирањето на овие особини, туку од овие, како што тој ги нарекува особини, подоцна ќе произлезат денес познатите принципи во Канбан методологијата. Според Anderson, поистоветувањето на овие 5 особини со терминот принципи прв го употребил Scoot W. Ambler [21].

Основни принципи на Канбан со софтверското инженерство се следните:

1. *Визуелизација на Работата-во-Тек (Workflow)*

   Овој принцип се однесува на користењето на Канбан табла. Оваа Канбан табла често претставува обична или електронска бела табла, со која имаме визуелна слика за работниот процес. Со неа добиваме визуелна репрезентација на тековниот процес, а со тоа имаме преглед на она што сме го сработиле и она што треба да го сработиме. Таблата е поделена во колони каде секоја колона претставува една фаза од процесот или листа. Секоја работна задача која треба да се сработи или пак е веќе сработена си има свое име, свој ID, рок за завршување на истата (ако постои). Како што напредува развојот на софтверскиот процес, секојдневно тимот за развој директно прави промени на канбан таблата со што се согледуваат разните застои, напредувања, тесни грла, а повисоките инстанци од организацијата се информираат редовно. Па токму од овде произлегува целта од користењето на една ваква табла, а таа е управувањето со процесот на софтверски развој да се насочи кон децентрализирање во однос на менаџерот и процесот сам по себе да се организира и предвидува. Што значи ова? На пример, не мора еден софтверски менаџер да му наложува креирање дополнителни функционалности на програмерот повеќе отколку што е нумеричкиот лимит поставен во заглавјето на колоната од табелата која се нарекува Програмирање. Од самата табела се гледаат сите податоци врз основа на кои се врши комуникацијата меѓу членовите на тимот. Една од разликите меѓу Канбан и останатите агилни методи, на пример Scrum, е тоа што Scrum акцентот го става на blocking and status issues (Кој што сè правел вчера и што размислуваат да прават денес), а Канбан се фокусира само на застои од повисоко ниво.

2. *Ограничување на Работата-во-Тек (WIP- Work-In-Progress)*

   При управување со некој проект, тимот за развој на софтвер има доделено задачи кои треба да ги сработи најчесто за некое дадено време со дадени ресурси. За секој проект, без разлика на неговата големина како и тимот кој работи на него, постои оптимален квантитет на работа во еден момент за конкретен процес кој притоа нема да влијае негативно на ефикасноста

односно постои граница, лимит на бројот на задачи на кои работиме и неговата вредност честопати е пониска од онаа која ја претпоставуваме. Овој лимит притоа нè попречува во успешното извршување на задачите. Со ограничување на Работата-во-Тек ние всушност имплементираме „pull“[18] систем на дел од или целокупната Работа-во-Тек. Токму Канбан е пример за таков „pull“ систем. Во секоја фаза од развојот на проектот, ограничена е Работата-во-Тек и нови задачи се повлекуваат (pulled) кон наредната фаза со оглед на капацитетот.

Со ограничување на Работата-во-Тек се намалува и тековното време на управување (lead time[19]), а ова придонесува за зголемување на квалитетот на сработените задачи и со тоа зголемување на целокупната продуктивност на тимот. За да можеме да го одредиме лимитот на Работата-во-Тек треба добро да ги препознаеме и разбереме застоите, блокадите во проектот и брзо да реагираме.

3. *Мерења и управување со Текот на податоци*

Текот (движењето) на работните задачи кои треба да се извршуваат во секоја од развојните фази треба да се надгледува и да се водат забелешки за истиот. Оваа активност се нарекува Мерење на текот (Measuring Flow). Во идеален случај овој тек треба да е брз и непречен. Брз и непречен тек значи брзо креирање на вредности од страна на нашиот систем, а со тоа намалување на ризикот и избегнување на цената која следи како резултат на задоцнувањата, а сето тоа во предвидливи рамки.

4. *Направи ги процесните полиси експлицитни*

Дефинирањето на полиси во рамките на Работата-во-Тек (WIP), приоритизацијата на работната листа (work queue), испорака на cadence[20], lead времето за дадени класи на сервиси се дел од активностите кои ги превзема еден Канбан тим. Полисите треба да се добро дефинирани

[18] Канбан системите потекнуваат од фамилија на пристапи позната како *pull* системи. (Подетално опишани во секцијата 2.4).
[19] Времето за кое клиентот ќе даде барање (креирање на работна задача) до моментот кога ќе ја добие истата односно пуштена во употреба (Подетално опишано во секцијата 2.6.1.).
[20] Поимот cadence се однесува на пристап кој претставува исполнување на обврските и доверливоста во еден Канбан систем. Временски ограничените итерации (time-box iterations) се пример за еден вид форма на овој пристап.

односно експлицитни и тие треба да ја одразуваат тековната ситуација во која се наоѓа тимот. Самиот процес на развој на софтвер или на ИТ операциите кои се одвиваат треба да е експлицитен и добро разбирлив односно треба да се има јасна претстава за тоа што се работи и како се постигнува работата, како се извршуваат задачите итн. Ова е со цел за подобро решавање на проблемите и подобрување на самиот процес.

5. Користење на Модели за препознавање на можностите за подобрувања (Подобрување на соработката)
6. Процесот кој еден Канбан тим го следи треба да ја одразува тековната ситуација и да разгледува можности за подобрувања. Една таква често користена техника во линеарниот софтверскиот развој е Мапирање на Вредносниот Стрим (Value Stream Mapping) која е воведена во книгите на Mary и Tom Poppendieck. Ова е всушност техника за моделирање на процеси која има за цел идентификација на активностите кои даваат вредности и времињата на чекање помеѓу овие активности. Со оваа техника имаме увид во процесот и во сè она што претставува безвредносна[21] активност, а со тоа и увид во можности за подобрувања. Тимот треба да има јасна претстава за работниот тек, подпроцесите кои се реализираат, лимитот на работата, ризиците и слично тогаш тие можат заеднички да работат на решавање на проблемите и да предложуваат подобрувања во системот. Во книгата на David Anderson, Kanban - Successful Evolutionary Change [20], предложени се три модели и тоа: Теоријата на ограничувања (The Theory of Constrains)[22], Теоријата на длабоко знаење (The Theory of Profound Knowledge)[23] и Линеарниот економскиот модел (Lean Economic Model)[24].

3.8 Примената на Канбан во развојот на софтвер

Кога станува збор за развојот на софтвер, во рамките на Канбан процесите не се користат сигнали (визуелни картички како во TPS) за

[21] Термин за означување на активностите кои не даваат вредност во конкретен процес (на пр: период на чекање додека да се случи наредната активност која дава вредност).
[22] Студија за варијациите и како тоа влијае на процесите
[23] Студија за „тесните грла“ (bottlenecks).
[24] Овој модел е базиран на концептот за „губиток“ (waste) или muda, mura, mun.

„извлекување“ (pull) на работа. Тука се користат работни задачи (*work items*) на кои е поделен еден софтверски процес. Честопати овој виртуелен сигнал кој е тригер за понатамошна акција се генерира од некоја софтверска апликација. Токму придавката „виртуелен“ до именката сигнал се користи за да се означи дека нема предавање на физичка картичка која се пренесува за означување на *лимитот* на Работата–Во–Тек (WIP). Постојат голем број на онлајн, десктоп или мобилни апликации како алатка за управување на еден канбан тим. Такви се, на пример: Lean Kit, Kanbanery, Kanbanize и многу други.

Во еден тим за развој на софтвер од особено значење е факторот време за испорака на софтвер. За разлика од еден традиционален тим за развој на софтвер кој испорачува дадени функционалности за даден временски период, Канбан тимот се обврзува на испорака на дадена единична функционалност. За ова потребно е да го предвидиме времето на управување и циклусното време.

Значајна карактеристика (величина) на еден канбан систем е неговата „пропусна моќ“, односно *throughput* и таа се однесува на капацитетот на еден тим. Оваа величина се поврзува со поимот *velocity*[25] која еден Scrum тим ја следи. Како што се намалува Работата–Во–Тек (WIP), така се зголемува „пропусна моќ“. Оваа величина може да се изведе од законот на Little опишан во секцијата 2.5.2 од каде:

$$Cycle\ time = \frac{Work - In - Progress}{Throughput} \text{ од овде } Throughput = \frac{Work - In - Progress}{Cycle\ time}$$

Со користењето на Канбан во управувањето со софтверските проекти се согледуваат следните погодности:

- Имплементацијата на Канбан методологијата е лесна бидејќи има мал број на принципи и правила кои треба тимот да ги следи;
- Бидејќи примената на оваа методологија не подразбира големи промени во тековниот процес каде се применува тоа, всушност, посочува на минимален ризик при нејзиното имплементирање, а тоа е од особено значење;

[25] Метод за одредување на ратата со која еден Scrum тим конзистентно испорачува бизнис вредности. Тоа е мерка за одредување на работата која тимот ја завршил за време од една итерација (спринт).

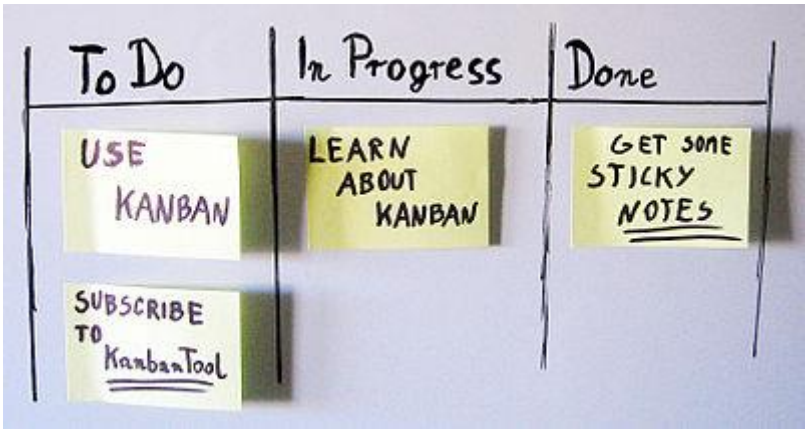
- Со визуелизацијата на софтверскиот процес на една табла се согледува тековната состојба и развојот на самиот софтверски процес. Ова значи и полесно согледување и решавање на проблемите и „тесните грла“ кои се појавуваат;
- Со користење на методологијата се мотивира и охрабрува тимот за понатамошно надградување и подобрување во работниот процес. Оваа особина произлегува од јапонскиот концепт ***Kaizen*** кој значи континуирано подобрување;
- Се зголемува корисничкото задоволство од крајните резултати;
- Зголемена е довербата кај сите засегнати интересентни групи;
- Се зголемува квалитетот на испорачаните продукти, се намалуваат трошоците итн.

3.9 Поим и видови на Канбан табли

Во агилниот софтверски развој пракса е да се користи визуелизација на тековниот проект и негово споделување со лепење на картички врз табла во некој работен простор. Овие табли се слични помеѓу себе, меѓутоа се разликуваат во зависност од применетата агилна методологија.

Во поширока смисла, станува збор за алатка со која се имплементира во канбан методот во некоја компанија за поддршка на производствените процеси. Во процесот на управување со софтверски проекти, Канбан таблата претставува табела со колони за секоја фаза од развојот на софтверскиот продукт. Имаме различни видови на вакви Канбан табли и истите се разликуваат во однос на: бројот и именувањето на колоните (фазите), портабилноста, намената итн.

На сликата подолу ни е прикажана наједноставната форма на една Канбан табла [24]. Оваа табла има само 3 колони и тоа: Да се направи (To Do), Во Тек (In Progress) и Завршено (Done).



Слика бр. 19 – Пример за едноставна Канбан табла

Најчестите примери на Канбан табли во линеарното и агилното софтверско инженерство се состојат од колоните: Backlog, Ready, Coding, Testing, Approval и Done. Покрај ова постои и друго именување на колоните, како на пр.: Next, In Development, Done, Customer Acceptance и Live.

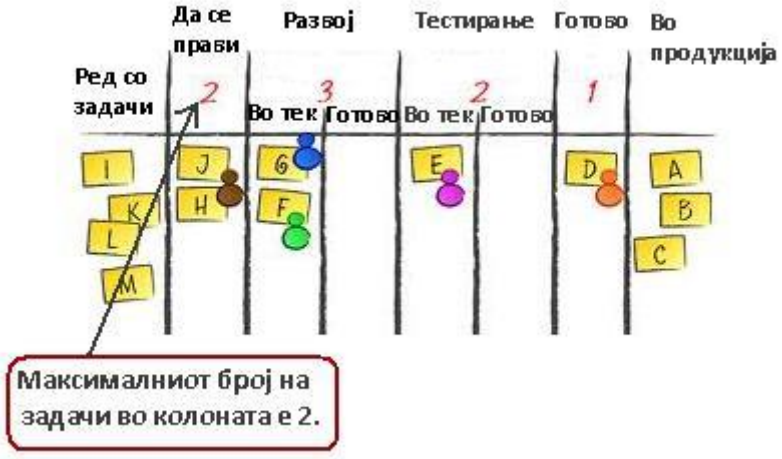
Работниот процес во рамките на еден Канбан тим е организиран во форма на задачи (work items) кои можат да бидат претставени во форма на лепливи ливчиња залепени на физичката табла, со името на лицето задолжено за нејзино извршување и со различна боја. Покрај овие ливчиња, можат да се користат и маркери за пишување на задачите во соодветните колони од таблата.

Покрај физичките и фиксни Канбан табли, постојат и *преносни, портабилни* табли уште наречени и *Kanban–nano* табли кои се претставени во форма на мала табла која членот од тимот може да ја носи на одредена локација. Овие табли биле користени во управувањето со еден проект во компанијата Central Computer Services. Во рамките на овој проект тимот работи во неколку помалку суб-тимови (вообичаени во пар). Кога еден суб–тим ќе извлече, „pulls“ корисничка приказна (User story), тие ја расчленуваат приказната на помали делови во форма на задачи и ги стават овие задачи на нивните портабилни табли.

Слика бр. 20 – Пример на преносна Канбан табла

Она што е значајно да се наведе е тоа дека оваа Канбан табла многу наликува на една типична Агилна табла. Меѓутоа, она што ја разликува една Канбан табла и е суштинска особина на самиот канбан систем е поимот за ***ограничување (лимит)***. Бидејќи времето како фактор е од големо значење за еден канбан тим и навремената испорака на софтверскиот производ е клучна, ограничувањето на работниот квантитет е неминовен. Овде постојат два вида на ограничување и тоа:

1. **Ограничувања во редовите на чекање (*Queue limits*)** – Со ова ограничување се извршува Just–In–Time концептот. Ова ограничување треба да биде доволно големо за да тимот има работа, но доволно мало за да не дојде до натрупување на редот со задачи кои предолго чекаат за да влезат во работниот процес.
2. **Ограничувања на Работата–Во–Тек (*WIP limits*)** – Со ова ограничување се одредува максималниот број на работни задачи кои можат да се наоѓаат во некоја од фазите (колоните на Канбан таблата) на развојниот процес. Тоа се броеви кои вообичаено се пишуваат над ознаката на колоната во таблата. Исто така, со ова ограничување се намалува истовременото извршување на активности (multitasking), потоа се зголемува пропусната моќ (throughput) и се мотивира тимот и тимската работа. На сликата подолу ни е прикажана табла со наведените нумерички ограничувања во една Канбан табла.



Слика бр. 21 – Пример на Канбан табла со означени ограничувања

4 Канбан Веб апликација (КанбанМАК)

4.1 Околина за развој и програмски јазици

За развојот на оваа апликација се користени неколку програмски јазици и PhpMyAdmin алатката за администрација на MySQL. Во однос на јазиците тука припаѓаат HTML, CSS,PHP, jQuery, AJAX итн. Од серверската страна се користи MySQL кој е најпознат систем за бази на податоци со најпознатиот структурен упитен јазик SQL заедно со PHP (Hypertext Preprocessor) скриптниот јазик за развој на веб во склоп на PhpMyAdmin.

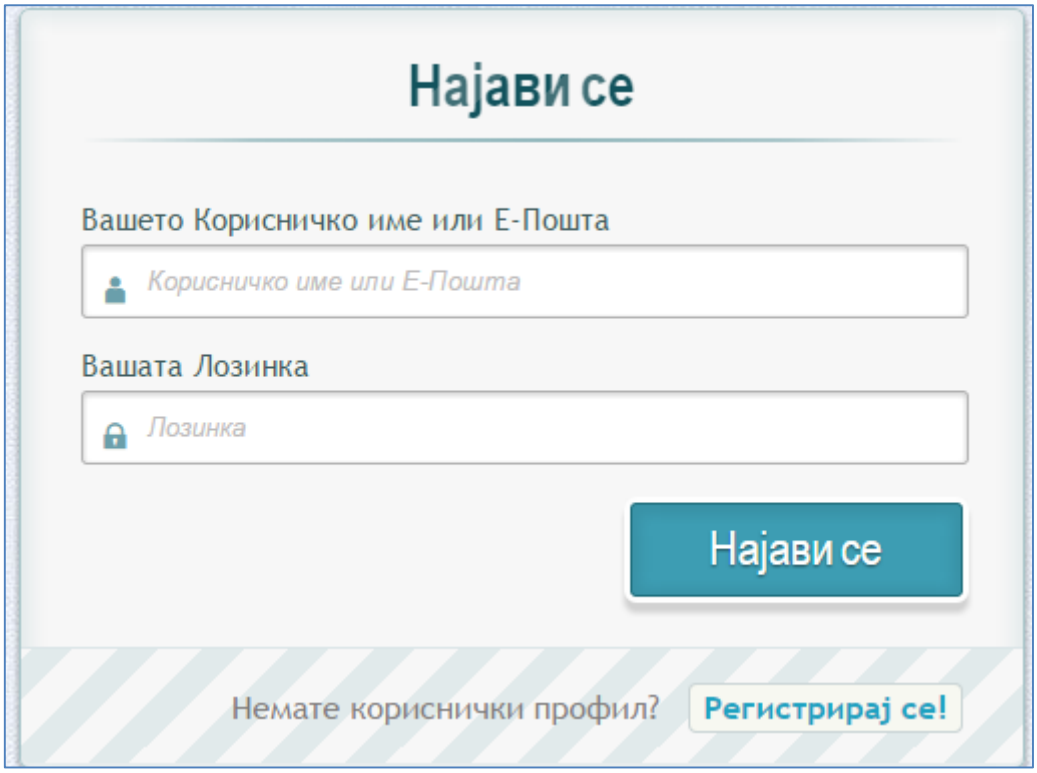
4.2 Опис на апликацијата КанбанМАК

Апликацијата КанбанМАК е веб апликација која се базира на дел од принципите на Канбан методологијата за развој на софтвер. Во однос на функционалностите кои ги нуди оваа апликација таа е поделена во неколку модули и тоа:

1. Модул за најава и регистрација;
2. Главна работна табла;
3. Модул за размена на пораки меѓу корисниците (public chat);
4. Модул за управување со картички (работни задачи);
5. Менаџирање со проектите;
6. Менаџирање со корисниците.

4.2.1 Модул за најава и регистрација

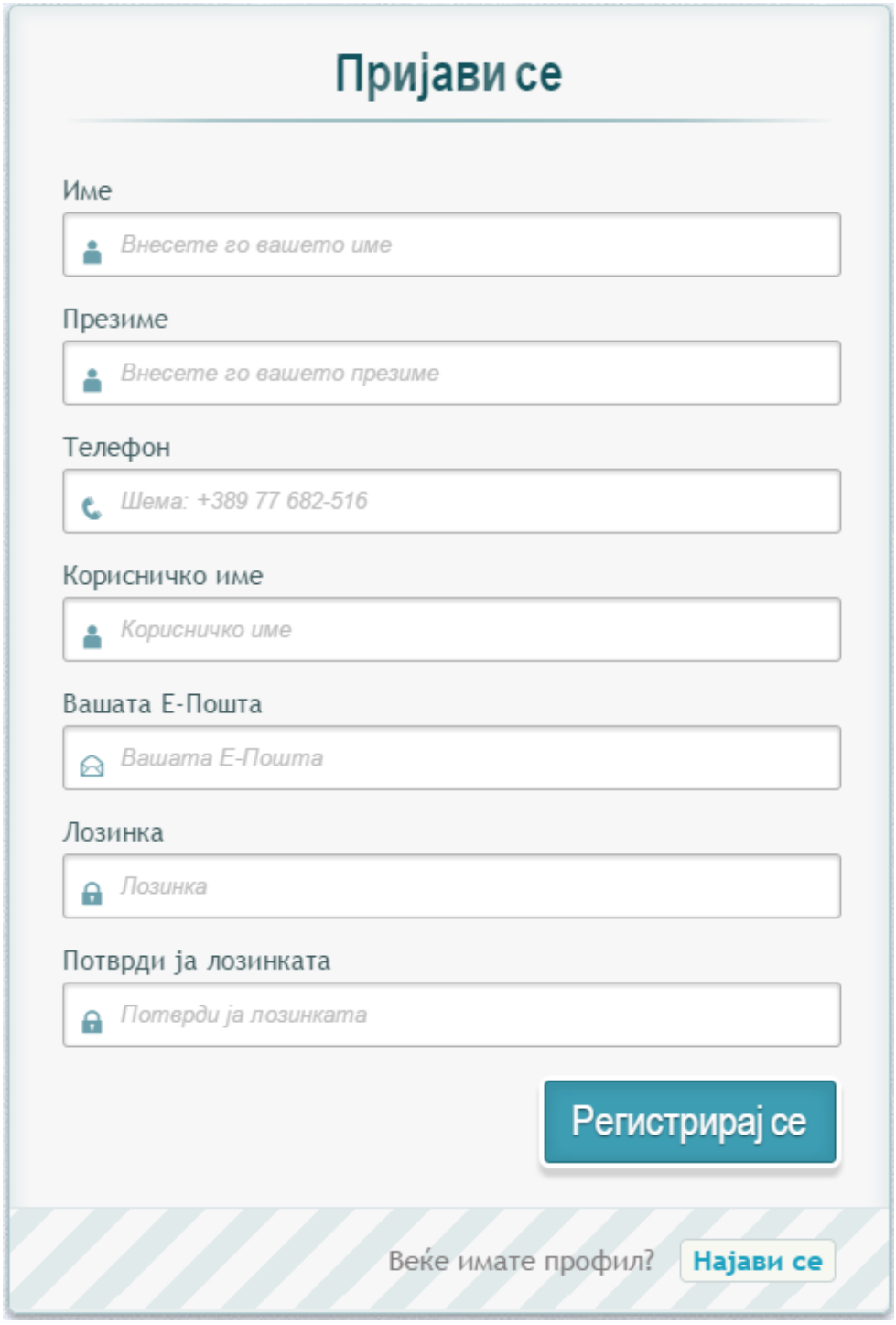
Почетната страна која се вчитува кога корисникот ја внесува адресата на КанбанМАК веб апликацијата е модулот за најава на веќе регистрираните корисници или пак доколку корисникот не е регистриран може да го направи тоа преку делот за регистрација кој се наоѓа како опција веднаш под формата за најава. На сликата подолу ни е прикажан модулот за регистрација на апликацијата.



Слика бр. 22 – Модул за најава на корисниците

Како што е прикажано на сликата, за да се најави корисникот потребно е да ги внесе своето корисничко име или електронска пошта и неговата лозинка. Ако точно се внесени овие податоци се појавува работната табла на корисникот и може да започне со работа. Ако корисникот го погреши своето корисничко име или е-пошта, се појавува порака дека таков корисник не постои и повторно се прикажува модулот за најава. Доколку корисникот ја погреши својата лозинка повторно се прикажува модулот за најава и се појавува порака за погрешна лозинка.

Модулот за регистрација е прикажан на наредната слика. Овој модул претставува форма со полиња кои новиот корисник ги внесува за да се регистрира. Сите полиња се задолжителни и имаат валидација доколку корисникот не внесе некое поле или пак внесе погрешен тип на податок за некое поле.

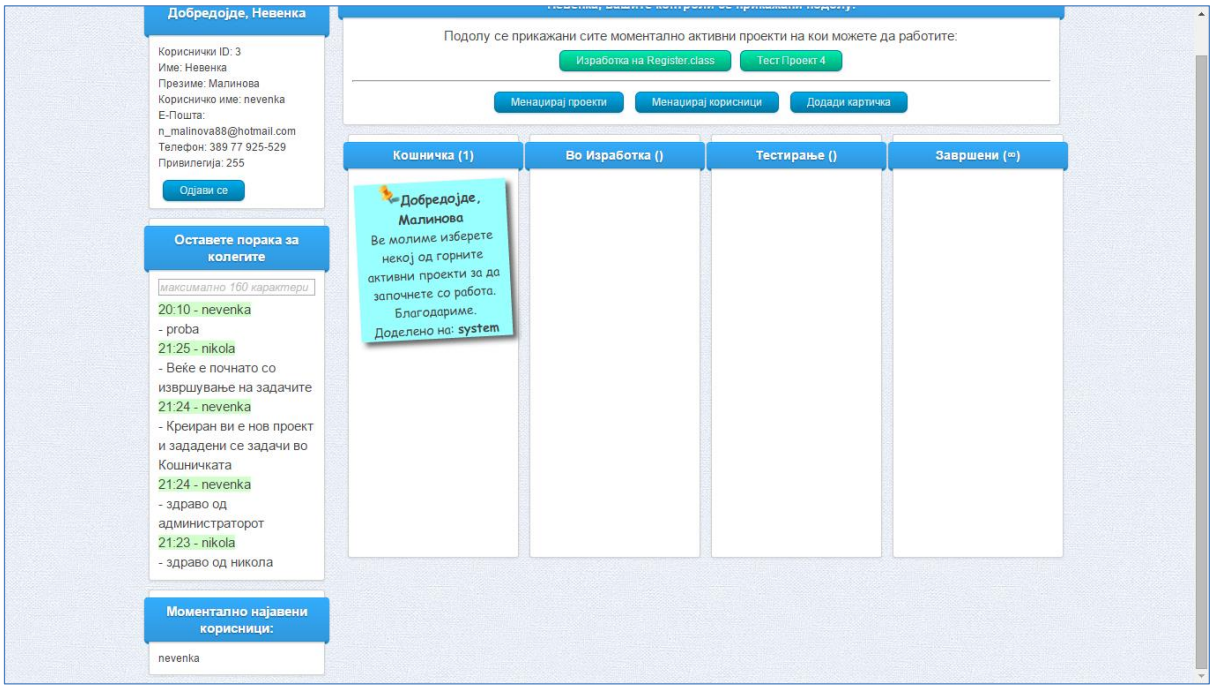


Слика бр. 23 – Модул за регистрација на корисниците

Кога корисникот ќе ги внесе правилно сите податоци тогаш се појавува порака дека корисникот е успешно регистриран. Сепак, за да може да започне со користење на оваа апликација, администраторот мора да му дозволи пристап, односно да му ја промени нултата (0) привилегија која автоматски им се доделува на новорегистрираните корисници.

4.2.2 Главна работна табла

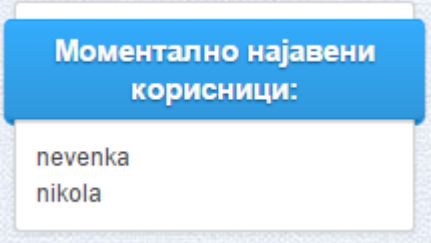
Работната табла е примарна околина за работа која е различна за секој корисник на апликацијата. Откако корисникот ќе се најави, неговото корисничко име се појавува во делот кај Моментално најавени корисници (Слика бр. 25) и се прикажува порака за добредојде која се наоѓа во колоната Кошничка. Ако корисникот има доделени проекти на кои работи тие ќе се појават во делот за доделени проекти. Додека корисникот не избере проект на кој ќе работи, не се прикажуваат лимитите во таблата ниту пак се појавуваат картичките за тој проект.



Слика бр. 24 – Работна табла на супер администратор

Во горниот лев агол се наоѓа панел со основните податоци за тековно најавениот корисник, како на пр. име, презиме, корисничко име, привилегија итн. Исто така, во овој дел се наоѓа и копче за одјава од апликацијата кое го води

корисникот на почетната страна за најава. Подолу се наоѓа четот за размена на пораките меѓу корисниците кој ќе биде подетално опишан во секцијата подолу.

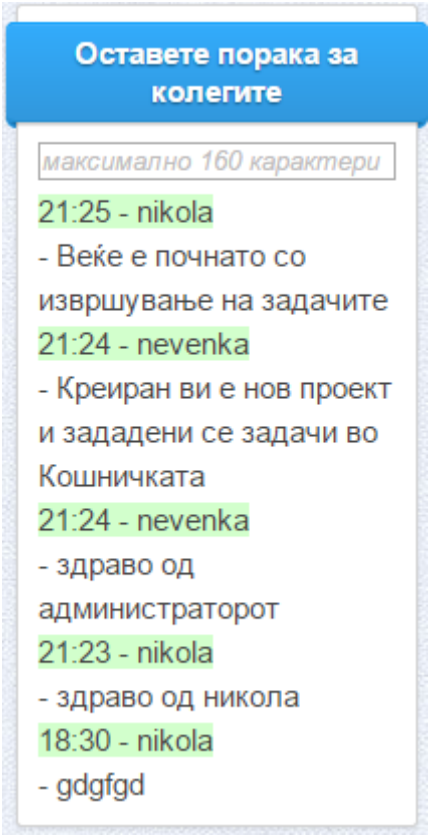


Слика бр. 25 – Приказ на тековно најавените корисници

Во централниот дел е самата работна табла која се состои од колони и картички. Колоните соодветствуваат на *фазите од еден софтверски развој* и тие се фиксни во апликацијата, а секоја картичка добива соодветна боја како што се поместува од колона во колона. Колоната е некоја *фаза од работниот процес (workflow)* во која се наоѓа картичката. Имињата на колоните се дефинирани интерно во самата апликација и нивниот број може да се менува, меѓутоа, треба да се направат промени во кодот за да се направи нивно динамичко именување и додавање/одземање. Над работната табла се наоѓаат активните проекти на најавениот корисник и модулите за управување со проекти и корисници доколку најавениот корисник е супер администратор.

4.2.3 Модул за размена на пораки меѓу корисниците (public chat)

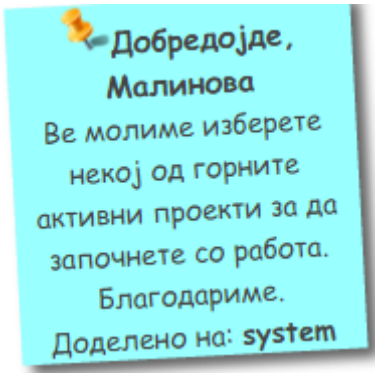
Овој модул се наоѓа во левиот панел од апликацијата и се користи за рамена на пораките меѓу корисниците (Слика бр. 26). Овој чет е од јавен карактер, што значи сите корисници можат да ги гледаат пораките и главно е наменет за размена на информации во контекст на користење на апликацијата и работните активности на корисниците. Како што е прикажано и на сликата, една порака која корисникот ја испраќа на четот има максимално 160 карактери (исто како и пораките кои ги испраќаме преку мобилните телефони). Доколку корисникот внесе повеќе од 160 карактери тогаш не се прикажуваат останатите карактери. Откако корисникот ќе притисне Ентер тогаш во панелот со пораките се појавува времето кога е пишана пораката заедно со корисничкото име одделени со средна црта.



Слика бр. 26 – Модул за размена на пораки (public chat)

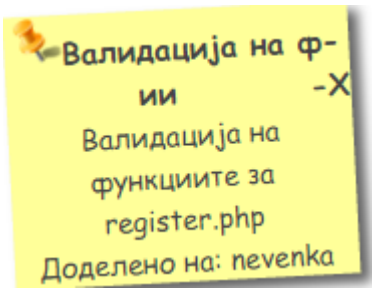
4.2.4 Модул за управување со картички (работни задачи)

Овој модул се однесува на самите картички со кои корисникот е во директна интеракција и основните функции поврзани со нив (додавање, поместување, бришење итн.).



Слика бр. 27 – Почетна картичка пред избор на проект

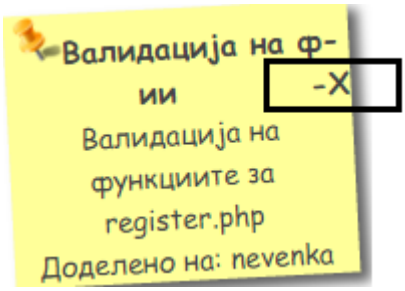
Картичката, или со друго име работна задача, е основна проектна единица која е носител на некоја проектна функционалност, особина или активност дефинирана од страна на администраторот. Доколку една картичка се наоѓа на работната табла тоа значи дека некој работи на неа. На сликата погоре ни е прикажан изгледот на картичката која се појавува после најавувањето на корисникот пред избор на проект за работа. Оваа картичка е системска и не е дел од ниту еден проект, туку само го информира најавениот корисник дека за да започне со работа треба да избере некој од проектите кои му се доделени. На сликата подолу ни е прикажана една картичка која е дел од активните проекти на еден корисник.



Слика бр. 28 – Приказ на картичка како дел од некој проект

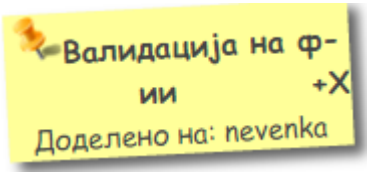
Секоја работна задача (картичка) ја има следната структура: наслов, опис и корисник на кој е доделена. Самата структура и дизајнот се креирани со цел да наликуваат на класичната физичка табла со стикери. За разлика од останатите Канбан веб апликации кои се достапни денес, како што се Kanban Tool, Kanbanize, Kanbanchi и многу други, КанбанМАК има многу едноставен интерфејс најмногу инспириран од класичната табла во однос на работната табла и картичките.

Од самиот приказ на картичката може да се забележи дека десно од насловот на картичката се наоѓаат две копчиња, како што е означено на сликата подолу. Копчето во форма на минус (**−**) се користи за пакување на описот на картичката. Истиот тој подоцна се претвора во (**+**) со што може повторно да се прикаже описот на картичката доколку корисникот го селектира.



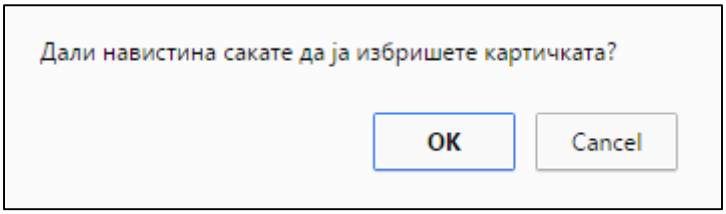
Слика бр. 29 – Копчиња за пакување/распакување и бришење

Минимизацијата на описот на картичката придонесува за подобра прегледност како на самата картичка, така и на работната табла. Секој проект има различен број на картички и тој зависи правопропорционално од големината на проектот, но претежно од критериумите за поделба на активностите на работни задачи. За оваа улога е одговорен самиот менаџер, а со оваа минимизација се постигнува позитивен ефект на прегледност дури и кога се работи за проект со поголем број на картички.



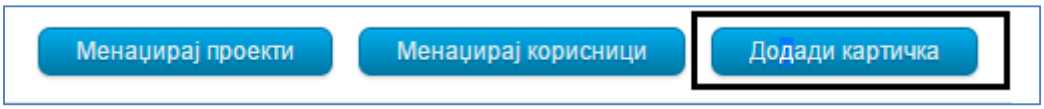
Слика бр. 30 – Картичка со минимизиран опис

До копчето за пакување/распакување се наоѓа копчето (***X***) кое се користи за бришење на картичката. Овие копчиња се појавуваат кај сите картички (освен кај почетната системска картичка) и се достапни за сите активни корисници на КанбанМАК апликацијата. Доколку корисникот сака да избрише некоја картичка треба само да кликне на ова копче. Меѓутоа, како и кај повеќето програмски активности поврзани со бришење, така и овде се појавува прозорец (JavaScript alert) за потврдување на бришењето (Слика бр. 31). Бришењето на картичката подразбира бришење од самата база, а со тоа и од проектот од кој таа е составен дел.



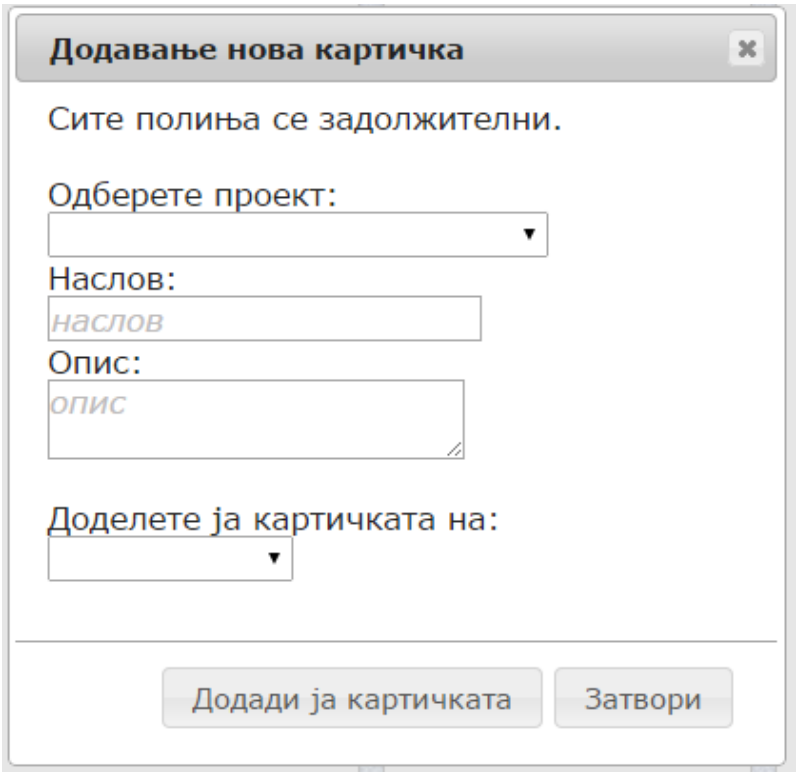
Слика бр. 31 – JavaScript alert за потврдување на бришењето

Наредната активност поврзана со картичките е нивното *креирање*. Опцијата за Додавање на картички се наоѓа под панелот со активни проекти веднаш до менаџирањето на корисници (Слика бр. 32). Додавањето на картички можат да го прават само администраторите и корисниците со најголема привилегија, односно супер администраторите.



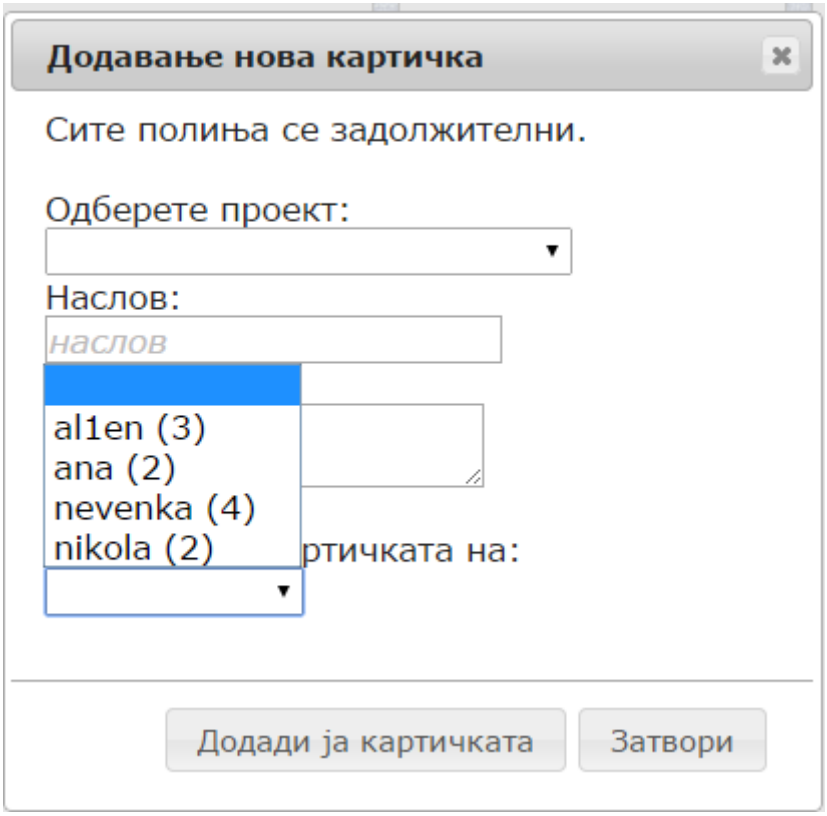
Слика бр. 32 – Додавање на нови картички

Откако корисникот ќе ја избере опцијата за додавање на картички се појавува дијалог прозорец прикажан на сликата подолу.



Слика бр. 33 – Дијалог прозорец за додавање на картички

Како што е прикажано на сликата, сите полиња мора да бидат пополенти. Најпрвин корисникот мора да избере проект на кој ќе припаѓа новокреираната картичка. Оваа листа на проекти (исто како и корисникот на кој се доделува) се пополнува динамички од базата на податоци преку AJAX повик. Потоа корисникот внесува наслов и опис на картичката. Откако корисникот ќе кликне на копчето Додади ја картичката, таа се додава во базата и се појавува во колоната Кошничка на тековниот проект. Овде мора да се напомене и дека при изборот на корисник на кого се доделува картичката веднаш до динамичката листа со имиња стои и една *нумеричка вредност во мали загради* (Слика бр. 34). Оваа вредност е од големо значење бидејќи таа го покажува бројот на картички на кои работат корисниците, па така самиот администратор има некоја визуелна претстава без да прави поединечна анализа за секој корисник.



Слика бр. 34 – Листа на корисници со број на тековни картички

Како што е и принцип на агилните методи, а конкретно на Канбан методологијата, еден Канбан тим акцентот го става на тековниот работен процес. Откако една работна задача (картичка) ќе се заврши, се оди на наредната картичка која се зема од колоната Кошничка. Од оваа листа

администраторот може да види кој од корисниците има најмногу/најмалку задачи и да одреди на кој од нив да ја додели задачата.

Последната активност поврзана со картичките е нивното поместување од колона во колона, што е директно поврзано со самиот континуален Канбан принцип. Нови картички се внесуваат во колоната Кошничка само доколку претходно менаџерот утврди дека доволен број на картички се завршени, односно се наоѓаат во колоната Завршени. Сите картички можат да се преместуваат сè додека не се надмине лимитот за дадена колона, што е подетално опишано во секцијата *3.4 Ограничување на Работата-Во-Тек.*

4.2.5 Менаџирање со проектите

Менаџирањето со проектите е достапно само за корисниците со најголема привилегија, односно за супер администраторите. Селектирањето на оваа опција го води корисникот во нова страна каде се прикажани сите податоци за проектите. Овие податоци се превземаат од базата на податоци и преку AJAX повик динамички се извршуваат операциите на додавање, промена и бришење на проектите. Овде се наоѓаат сите податоци кои се значајни за проектите како што се име, опис, статус, почеток и крај на проектот, како и лимитите за колоните од работната табла.

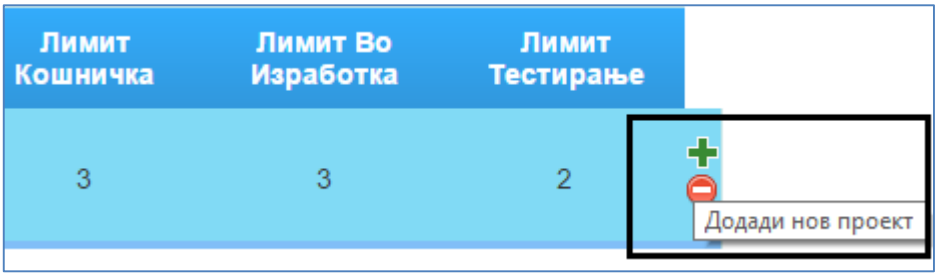
Табела за преглед на проектите и креирање на нови

Реден бр.	Име на проектот	Опис на проектот	Почеток на проектот	Крај на проектот	Статус на проектот	Лимит Кошничка	Лимит Во Изработка	Лимит Тестирање	
1	Изработка на Login.class	Ова е тест опис за проект со id 1.	2014/12/01	2014/12/31	1	3	3	2	
2	Изработка на Register.class	Ова е тест опис за проект со id 2.	2014/12/02	2014/12/30	1	2	2	3	
3	Канбан Табла	Ова е тест опис за проект со id 3.	2015/01/23	2015/02/05	0	6	2	4	
4	Тест Проект 4	Ова е тест опис за проект со id 4.	2015/01/08	2015/01/23	1	7	2	2	

Слика бр. 35 – Табела за додавање, промена и бришење на проектите

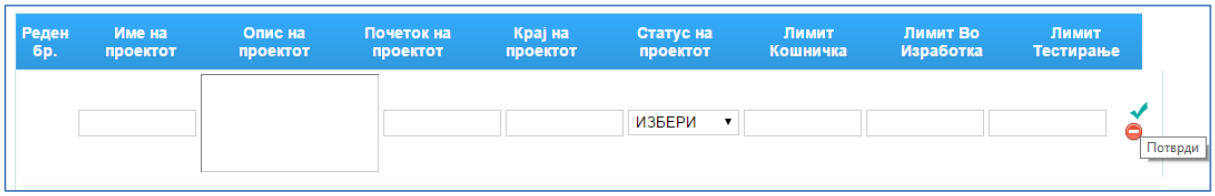
Најпрвин ќе ја разгледаме опцијата за *додавање на нови проекти*. Како што е прикажано на Слика 36, на крајот од секој ред од табелата се наоѓаат две копчиња: едното во форма на плус знак и другото во форма на стоп знакот. Како

што означува и самиот негов дизајн, копчето во форма на плус знак се користи за додавање на нови проекти.



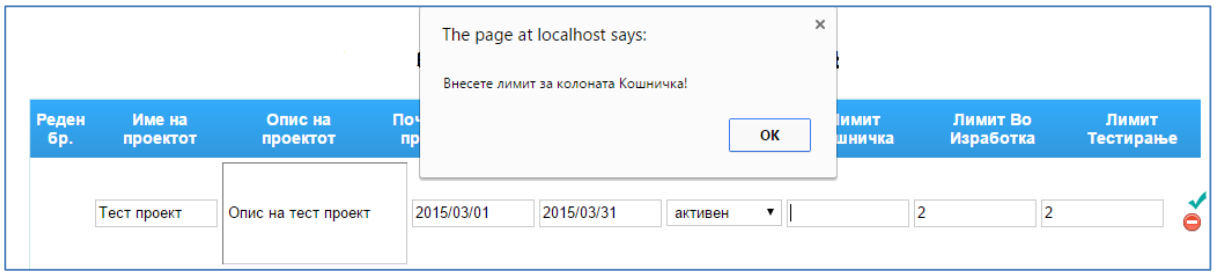
Слика бр. 36 – Додавање на проектите

Како што корисникот се приближува (on hover) кон копчињата, така се појавува и соодветен текст за намената на копчето. Корисникот може да кликне на кое било од копчињата означени со плус знакот (во кој било ред), со што секогаш на врвот од табелата ќе се појави нов ред со празни полиња кои корисникот треба да ги внесе за да креира нов проект (Слика бр. 37).



Слика бр. 37 – Додавање на нов проект

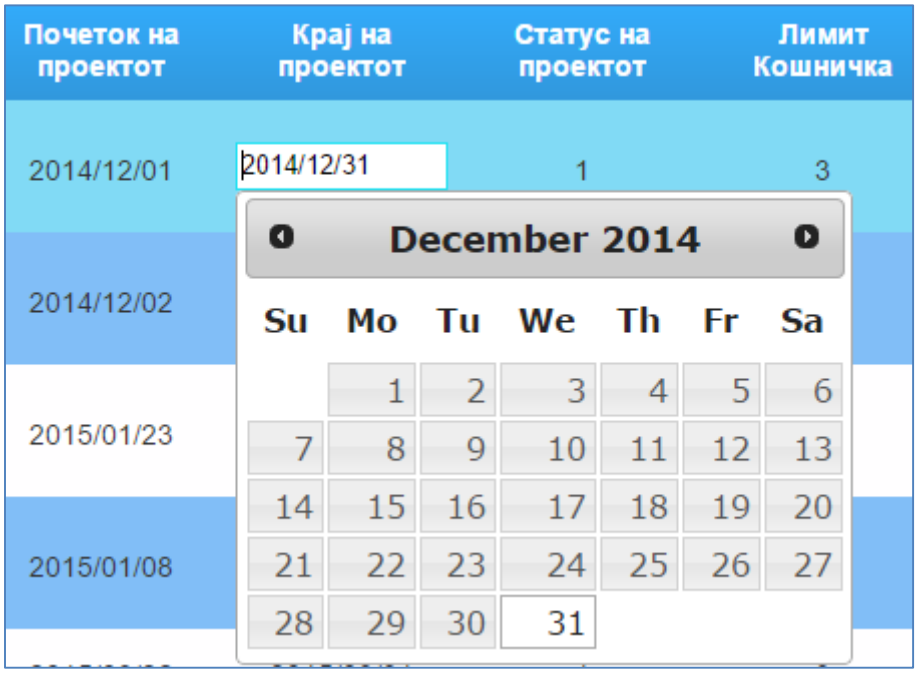
Сите полиња се задолжителни и за секое од дадените полиња постои валидација. Нов проект се додава само кога сите полиња ќе бидат соодветно пополетни. Ако корисникот не внесе некое од полињата, се појавува прозорец (Javascipt alert) за внесување на вредност во тоа поле.



Слика бр. 38 – Валидација на полињата за нови проекти

На Слика 38 е прикажано додавањето на еден нов проект каде сите полиња се пополнети освен полето за Лимит на кошничката. Кога корисникот кликнува на копчето Потврди , излегува прозорецот за известување за едно непополнето поле. Исто така, доколку корисникот се предомисли, може да кликне на копчето Откажи и да ја откаже операцијата на додавање.

Наредната опција достапна на администраторите во овој модул е промената на податоците за проектите. Администраторот ова може да го прави преку PhpMyAdmin алатката за администрација, но сепак, целта на овој модул е тоа да се направи преку самата апликација. Промената на податоците на проектите се прави едноставно со кликнување на полето кое сакаме да го промениме и потоа кликнуваме настрана или притискаме на тастерот Ентер од тастатура. Како што е прикажано на сликата подолу, корисникот го променува крајниот датум на проектот и со кликнување настрана автоматски ја активира оваа промена во базата без да мора да се користи PhpMyAdmin алатката.



Слика бр. 39 – Промената на податоците за проектите

Последната опција во овој модул е секако опцијата за бришење на проекти. Исто како и за додавањето на проекти, на крајот од секој ред во табелата за менаџирање на проекти се наоѓа копчето за бришење на проектите (Слика бр. 40).

Слика бр. 40 – Бришење на проектите

Операцијата бришење на проекти во апликацијата КанбанМАК се прави на едноставен начин преку овој модул. Но, во позадина, администраторот мора да направи детална анализа пред да го избриши самиот проект. Еден проект може да биде избришан во следните околности:

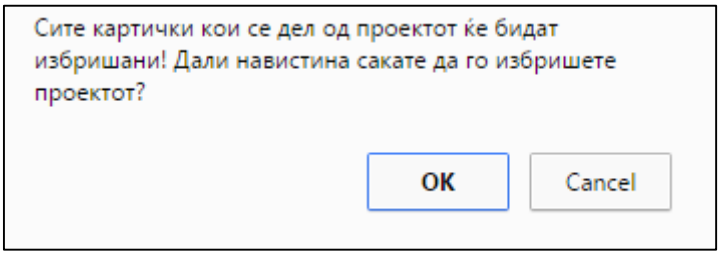
1. Проектот е *<u>завршен</u>* (сите картички се во колоната Завршени) и администраторот не сака повеќе тој проект да се појавува во базата на податоци и
2. Проектот е *<u>откажан</u>* поради каква било каква.

Секако ова може да се постигне и со промената на самиот статус на проектот од активен во неактивен (1->0), но така сè уште ќе е дел од базата на податоци, а со тоа и картичките кои се дел од проектот. Доколку администраторот сака да ги чува сите проекти во базата на податоци тогаш тој нема да ја користи опцијата за бришење на проектите, туку само ќе ги менува нивните статуси[26].

Бришењето на проектите исто така повлекува и друга значајна операција, а тоа е *бришење на картичките* кои се дел од тој проект. Затоа, откако корисникот ќе притисне на копчето за бришење на проектот се појавува прозорец (JavaScript alert) за потврдување на бришењето (Слика бр.41). Оваа операција

[26] Овде треба да се внимава на „неактивните“ картички кои се непотребен дел од базата (освен ако проектот не се реактивира).

се извршува автоматски без повторно вчитување на страницата. Алтернатива на оваа опција би била поединечното бришење на сите картички кои се дел од проектот, но ова е ризична и неефективна алтернатива бидејќи му одзема време и внимание на корисникот.

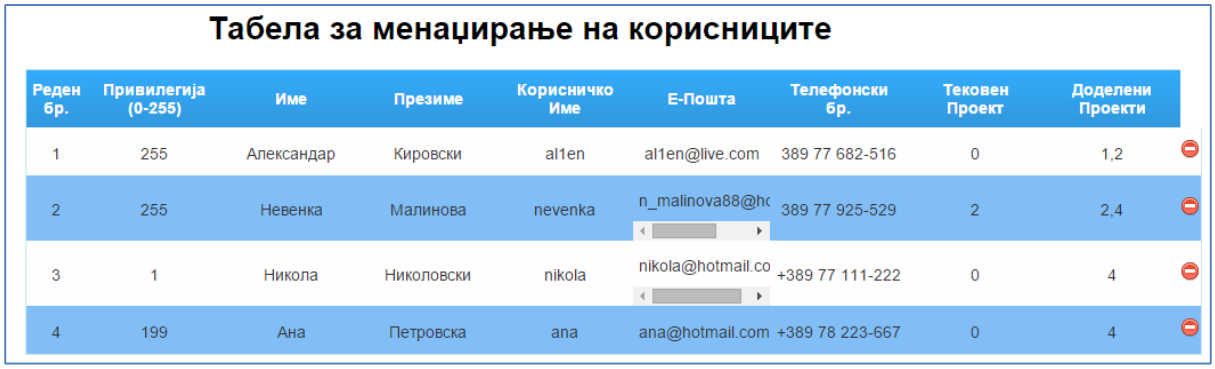


Слика бр. 41 – JavaScript alert за потврдување на бришењето

Бришењето на проектите како и промената на нивниот статус од активен во неактивен проект значи и бришење на тој проект од листата на активни проекти на најавениот корисник од работната табла.

4.2.6 Менаџирање со корисниците

Менаџирањето со корисниците е достапно само за корисниците со најголема привилегија, односно за супер администраторите исто како што беше случај и со менаџирањето на проектите. Исто како и за претходниот модул и овде податоците се превземаат од базата на податоци и преку AJAX повик динамички се извршуваат операциите на промена и бришење на корисниците. Овде се наоѓаат сите податоци за корисниците како што се име, презиме, привилегија, тековен проект на кој работи корисникот итн.



Слика бр. 42 – Табела за промена и бришење на корисниците[27]

За разлика од претходниот модул Менаџирање на проекти, овде немаме операција на додавање на нови корисници бидејќи нови корисници на апликацијата КанбанМАК се додаваат само преку модулот за регистрација. Затоа овде ќе бидат разгледани само операциите на промена на податоците за корисниците и нивното бришење.

Промената на податоците за корисниците се прави исто како и кај менаџирањето со проектите и тоа само со кликнување на полето кое сакаме да го промениме и потоа кликнуваме настрана или притискаме на тастерот Ентер од тастатура. Како што е прикажано на сликата подолу, администраторот прави промена на привилегијата на корисникот со име Никола со кликнување во полето Привилегија од табелата. Откако ќе ја внесе привилегијата за да се зачува промената во базата и на самиот кориснички профил администраторот само треба да кликне настрана или да притисне Ентер од тастатура. Ова автоматски поткренува промена во базата на податоци.

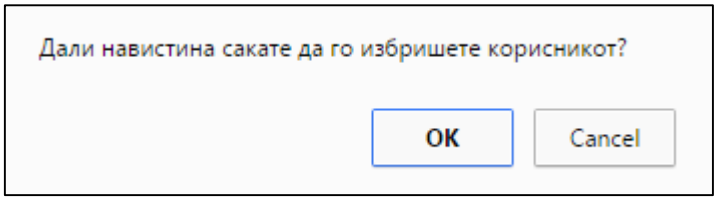
Реден бр.	Привилегија (0-255)	Име
1	255	Александар
2	255	Невенка
3	1	Никола
4	199	Ана

Слика бр. 43– Промена на податоци за корисниците

Исто како и за бришењето на проекти во претходниот модул и овде операцијата бришење на корисници се извршува со кликнување на копчето ⛔ кое се наоѓа на крајот од секој ред во табелата за менаџирање на корисници.

[27] Во колоната Тековен Проект и Доделени Проекти се наоѓаат само id –јата на проектите чии подетални информации корисникот ги добива од модулот за Менаџирање на Проекти или од базата.

Како и за бришењето картички и проекти и овде се појавува прозорец (JavaScript alert) за потврдување на бришењето (Слика бр. 44).



Слика бр. 44 – JavaScript alert за потврдување на бришењето

Бришењето на корисник подразбира бришење на сите податоци на корисникот од базата на податоци. Оваа операција *не е реверзибилна* и за да ја користи апликацијата треба повторно регистрирање на корисникот. Исто така постои и опција за деактивирање на корисниците со што тие ќе останат во базата на податоци, но нема да имаат пристап до апликацијата. Ова може да се постигне со промена на Привилегијата на корисникот во 0 со што добива ист статус како на новорегистриран корисник.

Овде мора да се напомене и за *сигурносниот аспект* (Security aspect) на модулите Менаџирање на корисници и Менаџирање на проекти. Со операциите на додавање, промена и бришење на проекти и корисници овие модули се во директна изложеност на напади за преземање податоци од базата како што се, на пример SQL Injections нападите (и многу други). Доколку не би постоела соодветна заштита во програмскиот код, неавторизираниот корисник преку овие напади ќе може да прави упити (queries) во базата, а со тоа да има пристап дури и на целата база на податоци. Затоа во овие модули секој пренос на податоци кој се прави *од база* и *кон база* е заштитен преку соодветни функции за енкриптирање[28] и декриптирање[29].

4.3 Типови на корисници во КанбанМАК апликацијата

Во однос на привилегијата која ја имаат корисниците при користење на оваа апликација имаме неколку типови корисници и тоа:

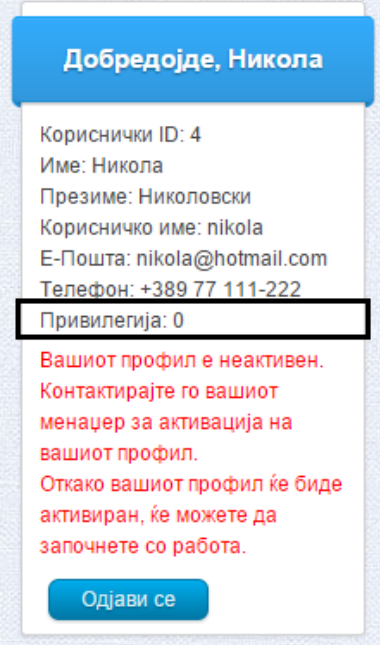
[28] Енкрипцијата се прави со php ф-јата strrev (податок) која го променува стрингот (податокот) во обратен редослед на што потоа неколку пати се повикува ф-јата base64_encode(..) која го енкриптира податокот со MIME база64.
[29] Декрипцијата се прави во обратен редослед од енкрипцијата и тоа најпрвин неколку пати се повикува ф-јата base64_dencode (податок) и потоа се повикува strrev(..) ф-јата.

1. Корисници со 0 привилегија (неактивирани корисници),
2. Корисници со привилегија од 1 до 199 (парцијални корисници),
3. Корисници со привилегија од 200 до 254 (администратори).
4. Корисници со 255 привилегија (Супер администратори)

4.3.1 Неактивирани корисници

Во првиот тип спаѓаат *новите корисници* кои се регистрирале на КанбанМАК апликацијата. За да започнат со користење на оваа апликација потребно е најпрвин одобрение од друг корисник со поголема привилегија, односно од администратор. Овој администратор кој воедно е и менаџер може да ја менува привилегијата на новите корисници и тоа може секвенцијално да им ја зголемува почнувајќи од Привилегија 1 или пак може веднаш да му додели најголема привилегија на новиот корисник.

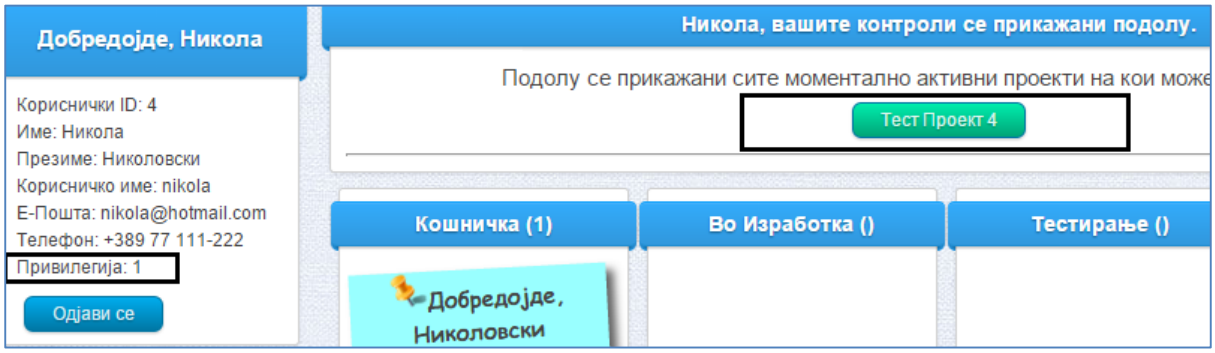
На сликата подолу ни е прикажан корисничкиот интерфејс на еден новорегистриран корисник кој се состои од една описна форма со копче за одјава на корисникот. На оваа форма се прикажани основните податоци за новиот корисник, како и известување означено со црвена боја на фонтот кое го информира новиот корисник за начинот на неговата активација. На сликата исто така е означена нултата привилегија. За да може Никола да започне со работа потребно е претходно супер администраторот да му ја промени привилегијата (минимална 1, а максимална 255). Иако е регистриран на веб апликацијата, сепак, новиот корисник не може да извршува никаква активност без дозвола од супер администратор.



Слика бр. 45 – Пример на неактивен профил

4.3.2 Парцијални корисници

Во вториот, пак, тип на корисници припаѓаат *парцијалните* корисници чија привилегија се движи од 1 до 199. Овие корисници имаат делумна контрола врз активностите кои можат да се извршуваат со помош на апликацијата КанбанМАК. Која било нумеричка вредност во рангот од 1 до 199 се смета како парцијална привилегија и истите активности се однесуваат за која било доделена вредност на корисникот. Така на пример, на сликата подолу ни е прикажан истиот корисник од претходната слика, но само со променета привилегија.



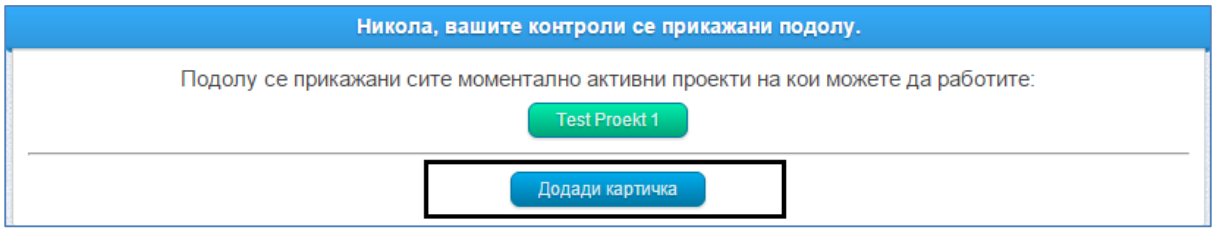
Слика бр. 46 – Пример на профил со парцијална привилегија

Како што е прикажано на сликата, Никола сега има веќе активен профил со можност за преглед на сите тековни проекти на кои работи. Релацијата меѓу

корисниците и проектите е следната: еден корисник може да работи на повеќе проекти, а секој проект се состои од еден или повеќе корисници. Доколку корисникот нема тековни активни проекти на кои работи, полето за активни проекти ќе биде празно. Исто така треба да се наведе дека опцијата за *Додавање на нови картички* не е достапна за парцијалните корисници.

4.3.3 Администратори

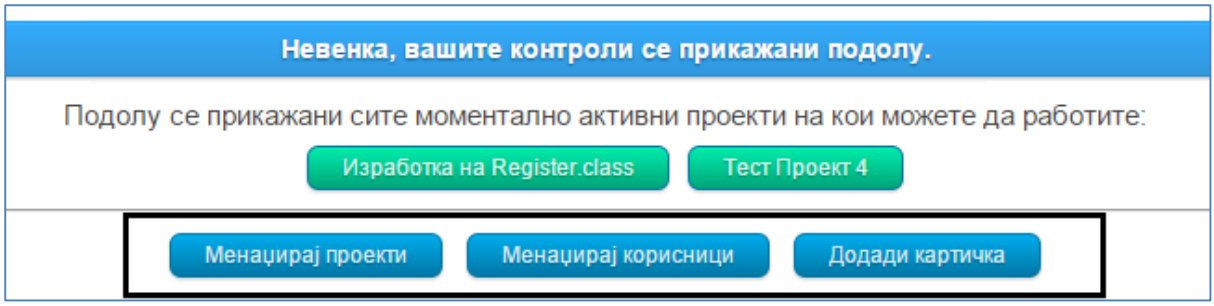
Администраторите се корисници кои имаат поголема привилегија од парцијалните корисници но помала од супер администраторите. Покрај тоа што тие имаат преглед на тековните проекти на кои работат, тие исто така за разлика од парцијалните корисници може да додаваат картички (работни задачи). Привилегијата на овие корисници се движи од 200 до 254. На сликата подолу ни е прикажан изглед на профил на еден администратор.



Слика бр. 47 – Пример на профил на администратор

4.3.4 Супер администратори

Супер администраторите, пак, се корисници со најголема привилегија на КанбанМАК апликацијата. Покрај приказ на тековите проекти, тие можат и да додаваат нови картички (работни задачи) во кој било од проектите, а исто така и да менаџираат со сите активности поврзани со проектите и корисниците. Менаџирањето со корисниците и проектите се однесува на претходно погоре опишаните модули. На сликата подолу ни е прикажано како изгледа профилот на еден супер администратор на оваа апликација.



Слика бр. 48 – Делумен изглед на профил на супер администратор

На оваа слика се означени главните модули, односно функционални делови од апликацијата кои се во одговорност на супер администраторот, но прикажани се и тековните проекти на кои работи. Супер администраторот може, но и не мора да има доделени проекти. Во нашиот случај имаме два проекти со имиња Изработка на Register.class и Тест Проект 4 на кои тековно работи супер администраторот со име Невенка. Доделувањето проекти се врши преку доделувањето на картички каде се избира за кој проект се креира картичката и за кој корисник. Така еден супер администратор може сам на себе да си додели проект, поточно работна задача или пак некој друг администратор со истата привилегија да му додели.

4.4 Ограничување на Работата-Во-Тек во КанбанМАК (WIP лимити)

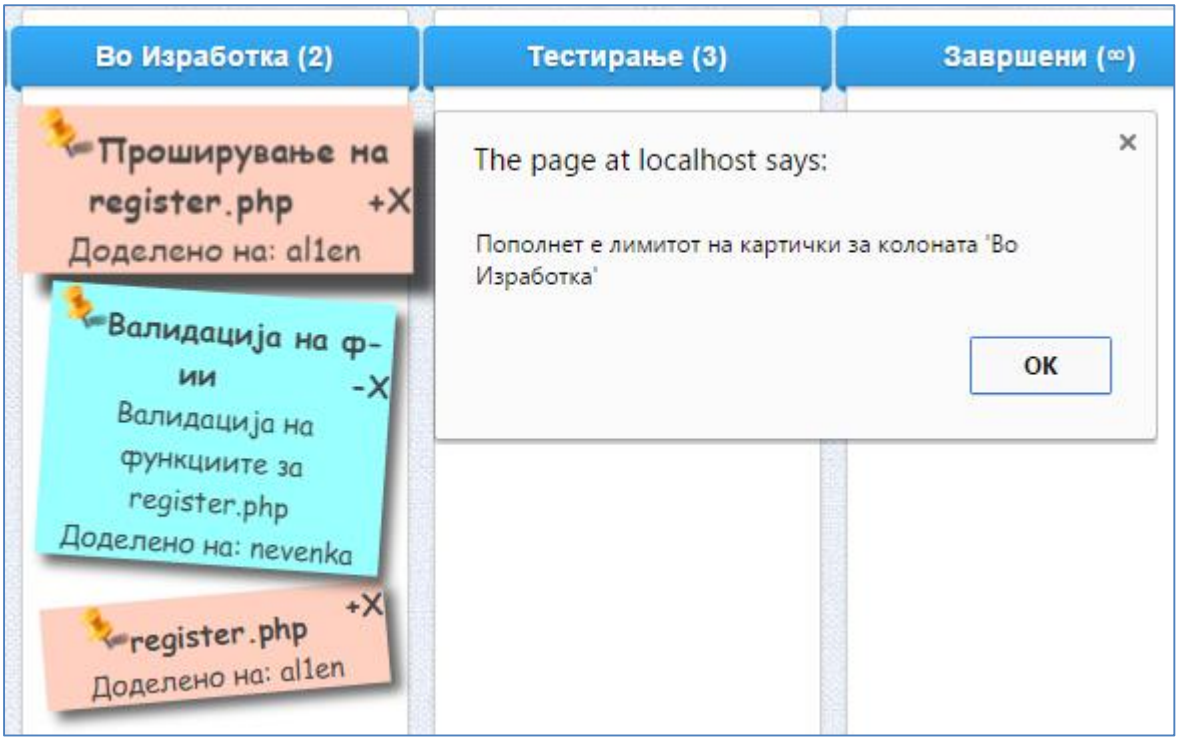
Главниот принцип на Канбан методологијата е Ограничување на Работата-Во-Тек (Work-In-Progress Limits). Овој принцип е многу значаен бидејќи тој, всушност, воведува ред во „хаосот од работни задачи“. Во суштина, има два вида на лимити: Ограничувања на редовите (Queue limit) и Ограничување на Работата-Во-Тек (WIP limits). Предмет на интерес овде се WIP лимитите. Кога не би постоеле лимитите во една Канбан табла таа би била само една обична Агилна табла. Овде, нормално, произлегуваат неколку прашања поврзани со дефинирањето и предностите од овој принцип, како на пр:

1. Што треба да имаме предвид кога ќе ги дефинираме лимитите на еден проект?
2. Како КанбанМАК апликацијата ги имплементира лимитите и дали еден Канбан систем е Канбан без овие лимити?
3. Дали промената на иницијално поставените лимити е индикатор за лоша проценка на менаџерот?

4. Како се согледуваат предностите од поставувањето на овие лимите?

Дефинирањето на лимитите во КанбанМАК апликацијата се прави при самото креирање на проектот каде за секоја колона се поставува нумеричка вредност која го претставува лимитот (Објаснето подетално во секцијата 3.2.5 Менаџирање со проектите: Додавање на проекти). За да ги поставиме овие лимити треба да знаеме колку активни корисници на апликацијата ќе бидат дел од тимот задолжен за даден проект и колку работни задачи (картички) ќе имаме во даден момент. Еден од пристапите при дефинирањето на лимитите е да започнеме со помали вредности и да ги зголемуваме. Во апликацијата КанбанМАК лимитите за даден проект се задолжителни полиња при креирањето на даден проект кои подоцна можат да се едитираат. Без овие полиња, т.е. без лимитите, Канбан таблата би била обична TO DO листа без притоа да имаме увид во *„тесните грла“* (bottlenecks), а без нивно идентификување немаме ни начин за нивно отстранување.

Во рамките на КанбанМАК апликацијата лимите на даден проект се наоѓаат десно од името на колоната во мали загради. Секој проект си има свои лимити и тие се ограничување на самата тековна работа. Како што корисникот селектира нови проекти, така се менуваат и картичките во таблата, а секако и самите лимите во колоните. Како што е прикажано на сликата подолу, доколку еден корисник сака да помести картичка во колона со веќе исполнет лимит се појавува прозорец, поточно JavaScript alert за надминување на лимитот. Поместената картичка автоматски се враќа назад во *претходната положба* откако ќе се појави прозорецот за предупредување. На ваков начин е имплементирано Ограничувањето на Работата-Во-Тек во КанбанМАК апликацијата.



Слика бр. 49 – Ограничување на Работата-Во-Тек (WIP limits) во КанбанМАК

Како што беше и претходно споменато, супер администраторот може да ги променува иницијално поставените лимити за даден проект со тек на време. Оваа промена не значи лоша проценка на самиот администратор при почетното дефинирање, туку потреба за прилагодување на самиот работен тек. Голема е веројатноста за корекција на овие иницијални вредности и нивна промена со тек на време.

Предностите од овие лимите се повеќе од очигледни. Корисникот во секој момент има увид на работниот капацитет: колку работа може да се сработи во даден момент, намалено е паралелното извршување на активности, поголема е предвидливоста во преценката за испорака, а со тоа е поголема и успешноста на тимот за развој.

5 Евалуација

Евалуацијата на една апликација е од особено значење за сите аспекти поврзани со самата апликација. Самата практична примена на апликацијата и повратниот одговор од корисниците е токму она што ја прави функционална апликацијата, со што се придонесува за нејзино понатамошно усовршување. Во

ова поглавје ни е претставена анализата на практична примена на апликацијата КанбанМАК како и добиените резултати од користењето на истата.

5.1 Анализа на користењето на апликацијата КанбанМАК во рамките на одделението за развој на една македонска ИТ компанија

По имплементацијата на решението, истото се претстави на вработените во компанијата, преку пишана документација и презентации. Истите имаа период на тест од 14 денови, по што дадоа низа сугестии и забелешки за подобрување на апликативното решение кои беа дополнително имплементирани.

По воведувањето на апликативното решение следеше период на евалуација на корисноста на апликацијата во секојдневното работење на компанијата. Истата беше тестирана на проекти од средна големина/комплексност, т.е. проекти со времетраење од 2-6 месеци во кои учествуваат 1-10 вработени истовремено. Самата апликација беше ставена во функција во период од 14 месеци и беше користена при изработката на 23 проекти на компанијата за време на периодот на евалуација.

5.2 Цел, методологија и опфат на анализата

Целта на анализата беше да се добијат сознанија за нивото на користењето на апликацијата, како и за придобивките од користењето на решението. Анализата се базира на два методолошки пристапи и тоа:

- Автоматска анализа на базата на податоци и
- Анкета со вработените.

Со цел да се добие јасна претстава за користењето на апликацијата беше спроведена веб анкета меѓу вработените од ИТ компанијата.

На анкетата беше побаран одговор од вработените во компанијата (вкупно 45 испитаници), при што одговор дадоа 88,8% од вработените лица.

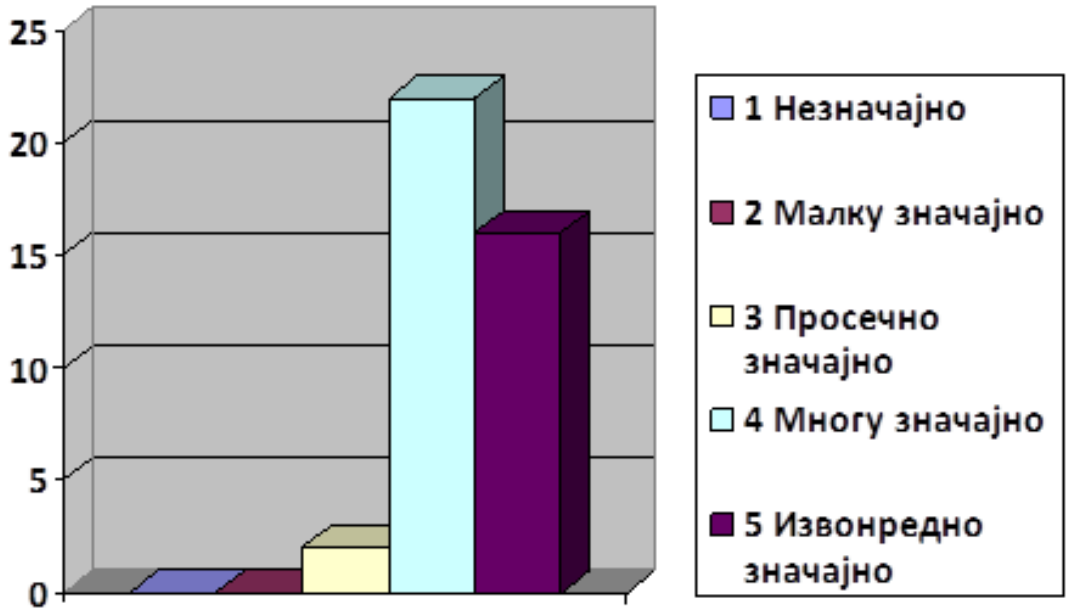
5.3 Резултати од анализата

На соодветните прашања се добија одговори од испитаниците, при што се добиени сублимирани резултати кои се прикажани во табеларен и во графички облик.

Прашањата и приказите се:

Табела бр. 3 – Прашање со избор на еден одговор

Дали сметате дека е постигнато значајно подобрување на работењето со воведувањето на апликацијата ?	*Одговори*
1 - незначајно	0
2 - малку значајно	0
3 - просечно значајно	2
4 - многу значајно	22
5 - извонредно значајно	16

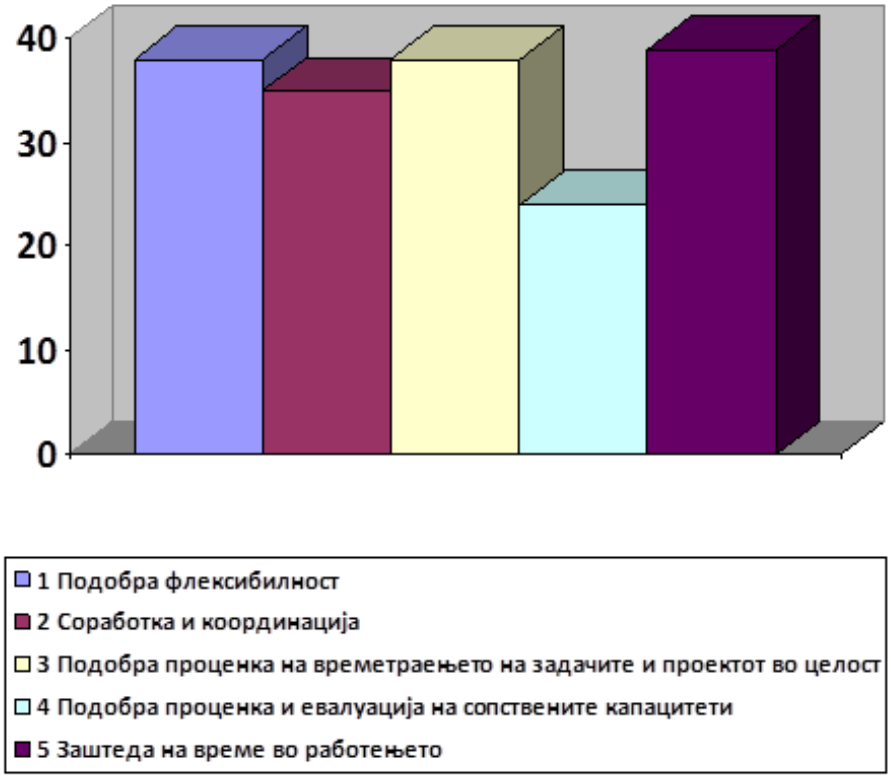


Слика бр. 50 – Столбест 3Д графикон за Табела бр. 3

Од Табелата бр. 3 и визуелно од Графиконот прикажан на Слика бр. 50 се гледа дека повеќе од половина од анкетирани вработени дале оценка 4 за подобрувањето на работењето по користењето на апликацијата. Исто така, 40%, односно 16 вработени дале и највисока оценка, што е многу задоволителен податок.

Табела бр. 4 – Прашање со повеќе одговори

Дали со користењето на апликацијата, добивате некоја од следниве предности?	*Одговори*
1 - подобра флексибилност	38
2 - соработка и координација	35
3 - подобра проценка на времетраењето на задачите и проектот во целост	38
4 - подобра проценка и евалуација на сопствените капацитети	24
5 - заштеда на време во работењето	39

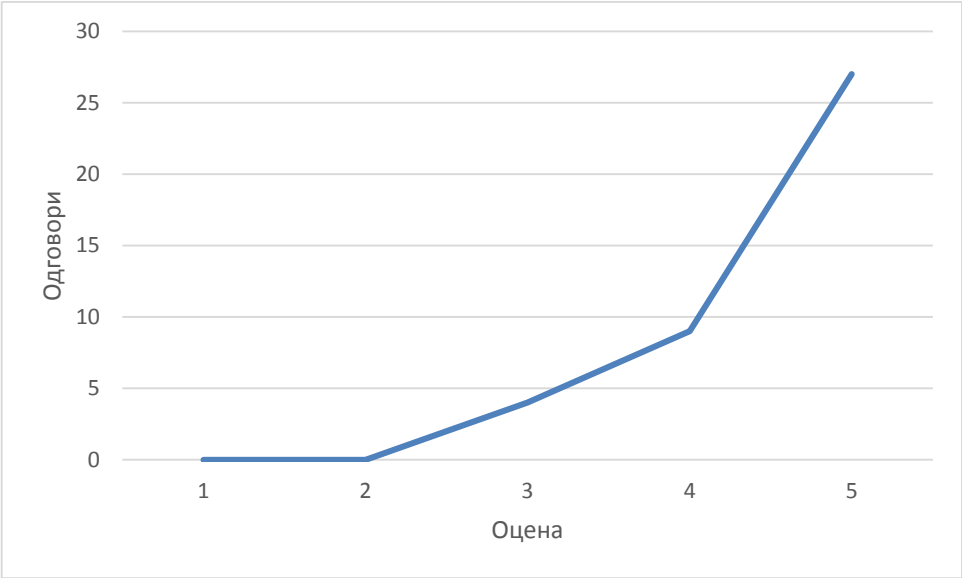


Слика бр. 51 – Столбест 3Д графикон за Табела бр. 4

Од Графиконот пак, прикажан на Слика бр. 51, се согледува дека вработените дале одговор на голем дел од наведени предностите кои произлегуваат со користење на апликацијата.

Табела бр. 5 – Прашање со избор на еден одговор

Како би го оцениле квалитетот на понудените функционалности, со скала од 1 до 5 (1 - многу лош, 5 - многу добар)?	*Одговори*
1	0
2	0
3	4
4	9
5	27

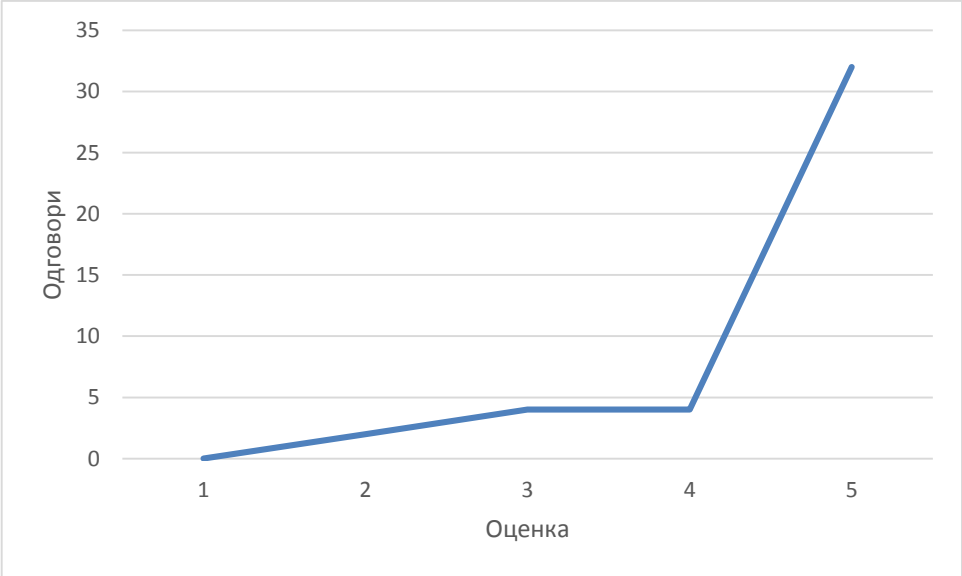


Слика бр. 52 – Линиски графикон за Табела бр. 5

Од Графиконот прикажан на Слика бр. 52 се согледува дека најголем број од вработените (поточно 67.5%) дале најдобра оценка за квалитетот на понудените функционалности, додека ниту еден од нив не дал оценки 1 и 2 за нив.

Табела бр. 6 – Прашање со избор на еден одговор

Како би ја оцениле навигацијата низ самата апликација и едноставноста за користење, со скала од 1 до 5 (1 - многу лош, 5 - многу добар)?	*Одговори*
1	0
2	2
3	4
4	4
5	32

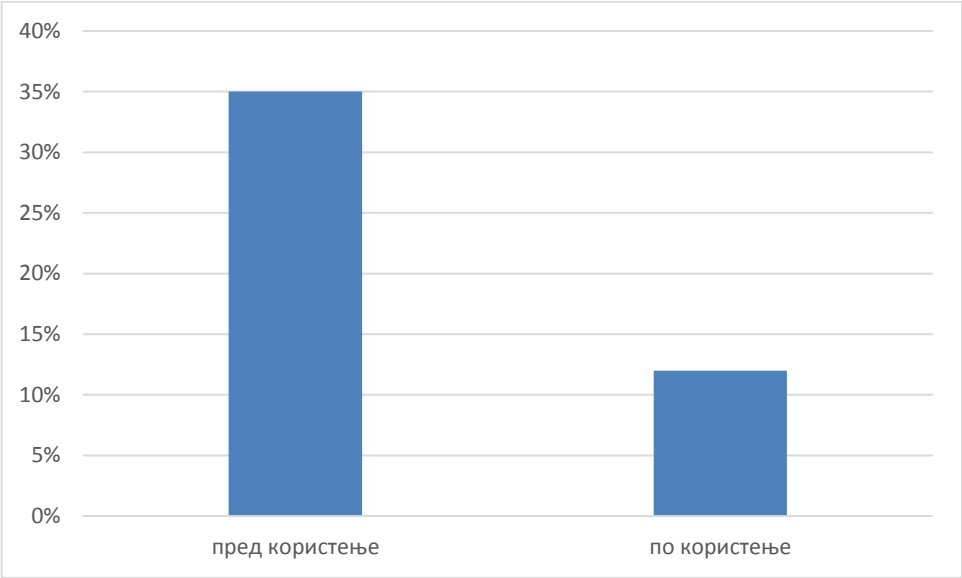


Слика бр. 53 – Линиски графикон за Табела бр. 6

Од Графиконот прикажан на Сликата бр. 53 се согледува дека 32 вработени, односно 80% од анкетираните вработени дале најдобра оценка за навигацијата на апликацијата и едноставноста на користење.

Табела бр. 7 – Проценето наспроти реално време пред и по користење на апликацијата

Процентуална разлика меѓу проценетото и реалното времетраење на задачите во проектите со средна големина (2-6 месеци)	*Одговори*
Пред употреба на апликацијата и методологијата Канбан	35%
По употреба на апликацијата и методологијата Канбан	12%



Слика бр. 54 – Процентуален столбест графикон за Табела бр. 6

Од Графиконот прикажан на Сликата бр. 54 се согледува намалување до 23% на процентуалната разлика на предвиденото и реалното времетраење на задачите по користењето на апликацијата. Намалувањето на оваа разлика е од големо значење бидејќи таа укажува на ефективноста на самата апликација КанбанМАК од аспект на навременото извршување на проектните задачи.

6 Заклучок

Софтверскиот продукт (програма, апликација) се разликува во однос на останатите продукти од другите науки на различни начини. Едни од карактеристиките на софтверскиот продукт кои воедно се и дистинкција од останатите производи се: развојот на еден софтверски продукт е *логичка и*

рационална работа за разлика од некоја друга дисциплина како што е, на пример, градежништвото, каде имаме физичка работа при изградба на некој објект. Друга разлика е тоа што развојот на еден софтверски продукт е неопиплив и не можеме да видиме колкав дел од работата е сработен. Трета разлика е неможноста да се измери комплексноста на еден софтверски продукт сè додека не се започне со негова изработка. Покрај овие постојат и други разлики. Иако моделот на водопад сè уште се користи и е една од најстарите методологии за развој на софтвер, сепак, докажано е дека потребата од нови методи е повеќе од потребна. Тоа беше причина за воведување на агилните методи како подобрувања на тогашните методи за софтверски развој.

Секој софтверски развој е приказна сама за себе и не постои таков модел во светот кој би се користел како единичен модел за сите типови на софтверски проекти. Кога станува збор за користењето на агилните методи, тимот одбира соодветен модел во зависност од типот на проектот и низа други критериуми. Сите агилни методи си имаат свои предности и недостатоци. Она што ја издвојува Канбан методологијата од останатите агилни методологии е фактот што таа е флексибилна во најголема мера и нема потреба од големи промени во тековните процеси при нејзиното имплементирање. Меѓутоа, овде секако постојат правила кои мора да се имплементираат. Со Канбан методологијата акцентот се става врз успешноста на изработката на софтверскиот продукт.

Една од главните предности на КанбанМАК апликацијата е *едноставниот кориснички интерфејс*. Ова се однесува како на самата работа табла, така и на самото менаџирање со проектите и корисниците. Кога станува збор за работната табла, избран е дизајн со цел што е можно повеќе да наликува на класичната табла на која тимот прикачува работни задачи во форма на стикер ливчиња. Ограничувањето на Работата-Во-Тек во апликацијата дава *визуелна контрола* на самото циклусно време, но и *идентификација* на „тесните грла“.

Навремената испорака на еден софтверски продукт е една од најважните карактеристики со кои треба да се одликува продукт. Квалитетот на еден софтверски продукт е секако повеќе од важна. Со успешните софтверски проекти се дефинира самата „*пропусност*“ на еден тим и се создава една база на искуство и знаење. Пропусноста е всушност она што погоре го дефиниравме

како *пропусна моќ* и се однесува на ратата на испорака на функционалности на клиентот.

За да имаме успешно спроведување на Канбан методологијата треба да се стави акцент на подобрување на факторите кои влијаат на пропусната моќ: циклусното време и Работата–Во–Тек. Подобрувањето на циклусното време и ограничување на Работата–Во–Тек придонесуваат за согледување на придобивките од оваа методологија.

Канбан методологијата нуди посебна метрика за надгледување и проучување на Канбан системот и тоа во стохастички рамки. Ова ни овозможува да ја прошириме довербата и да го предвидиме натамошниот развој. Зголемување на довербата значи зголемена соработка како во рамките на тимот, така и со бизнис околината, а воедно и наметнува промени за понатамошно подобрување и надградување.

7 Користена литература:

[1] Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Humt, A., Je®ries, R., Kern, J., Marick, B., Martin, R.C., Mellor, S., Schwaber, K., Sutherland, J., Thom, D.: Manifesto for agile software development. Website (2001) http://agilemanifesto.org/.;

[2] Jeffries R; Anderson A.; Hendrickson C.: (2000) *Extreme Programming Installed,* Addison - Wesley Professional.;

[3] http://en.wikipedia.org/wiki/Chrysler_Comprehensive_Compensation_System (Пристапено на 05.01.2013);

[4] Beck, K. (1999). *Extreme programming explained: Embrace change*. Addison-Wesley Professional;

[5] Cockburn, A. (2004). *Crystal "clear": A human-powered software development methodology for small teams*. Addison-Wesley Professional;

[6] Schwaber, K., Beedle, M.: „*Agile Software Development with SCRUM*", Prentice-Hall, 2002;

[7] Feature Driven Development (FDD) and Agile Modeling, Scott W. Amber & Associates, http://www.agilemodeling.com/essays/fdd.htm (Пристапено на 10.01.2013);

[8] Coad, P.,Luca, J.,Lefebyre, E. : *Java Modeling In Color with UML*: *Enterprise Components and Process*, 1999;

[9] Highsmith, J.: „Adaptive Software Development: A Collaborative Approach to Managing Complex Systems", New York, NY: Dorset House, 2000;

[10] Agile Project Management: Adaptive Software Development, http://www.adaptivesd.com , (Пристапено на 22.01.2013);

[11] Norvig P., Cohn D.: Adaptive Software, http://norvig.com/adapaper-pcai.html (Пристапено на 01.02.2013);

[12] DSDM Конзорциум, http://www.dsdm.org (Пристапено на 03.02.2013).

[13] Stapleton, J.: DSDM: The Method in Practice, Second ed", Addison Wesley Longman, 2003;

[14] Just-In-Time (JIT) Белешки од предавања. Future Undergraduates Ashland University. http://personal.ashland.edu/~rjacobs/m503jit.html (Пристапено на 22.12.2012);

[15] Gross, John M., Kenneth R. McInnis. *Kanban Made Simple: Demystifying and Applying Toyota's Legendary Manufacturing Process*. New York: AMACOM, 2003;

[16] Опис на Канан со примена на JIT http://www.brighthubpm.com/monitoring-projects/70336-kanban-explained-as-applied-to-jit/ (Пристапено на 05.12.2012);

[17] Taiichi O. Toyota Production System: *Beyond Large - Scale Production,* (1988);

[18] Just-In-Time - Филозофија на целосна елиминација на губиток http://www.toyota-global.com (Пристапено на 25.12.2012);

[19] Kanban Applied to Software Development: from Agile to Lean http://www.infoq.com/articles/hiranabe-lean-agile-kanban (Пристапено на 27.12.2012);

[20] David J. Anderson. (2003): *Kanban*: *Successful Evolutionary Change for Your Technology Busines*);

[21] Introduction to Kanban: http://www.drdobbs.com/architecture-and-design/introduction-to-kanban/225702051 (Пристапено на 24.12.2012);

[22] Poppendieck M., Poppendieck T.: "*Lean Software Development*", 2003 Addison-Wesley;

[23] Poppendieck M., Poppendieck T.: *Implementing Lean Software Development*", 2006 Addison-Wesley;

[24] H. Kniberg, M. Skarin: *Kanban and Scrum making the most of both*. C4Media, Publisher of InfoQ.com, USA (2010);

[25] *Ahmad, M. O., Markkula, J., & Oivo, M. (2013, September). Kanban in software development: A systematic literature review. In Software Engineering and Advanced Applications (SEAA), 2013 39th EUROMICRO Conference on (pp. 9-16). IEEE;*

[26] *Ahmad, M. O., Markkula, J., Oivo, M. & Kuvaja, P. (2014). Usage of Kanban in software companies: An empirical study on motivation, benefits and challenges (ICSEA), 2014: The Ninth International Conference on Software Engineering Advances;*

[27] *Cloud Software Finland, General Brochure. [Online] Available from: http://www.n4s.fi/en/ , http://www.cloudsoftwareprogram.org/general-brochure 2014.05.13*;

[28] *D. Anderson, Kanban – Successful Evolutionary Change for Your Technology Business. Blue Hole Press, 2010;*

[29] *M. O. Ahmad, K. Liukkunen, J. Markkula, "Student perceptions and attitudes towards the software factory as a learning environment," In Proceedings of IEEE EDUCON, 2014, pp. 422,428.*