УНИВЕРЗИТЕТ У БЕОГРАДУ

МАШИНСКИ ФАКУЛТЕТ

Горан П. Ђурић

Методологија оптимизације ефеката протока података на бази интеграције структурне системске анализе и ризика

докторска дисертација

Београд, 2020

UNIVERSITY OF BELGRADE

FACULTY OF MECHANICAL ENGINEERING

Goran P. Đurić

Methodology of optimization of data flow effects based on integration structural systems analysis and risk

Doctoral Dissertation

Belgrade, 2020

Комисија за оцену и одбрану:

Ментор:

др Часлав Митровић, редовни професор

Универзитет у Београду - Машински факултет

Чланови комисије:

др Александар Бенгин, редовни професор

Универзитет у Београду - Машински факултет

др Мирјана Мисита, редовни професор

Универзитет у Београду - Машински факултет

др Данијела Тадић, редовни професор

Универзитет у Крагујевцу - Факултет инжењерских наука

др Горан Воротовић, доцент

Универзитет у Београду - Машински факултет

Датум одбране докторске дисертације:

__.__. 2020.

РЕЗИМЕ

У докторској дисертацији истраживан је проблем оптимизације ефеката протока података код информационих система и развој методологије за ефикасно управљања процесима протока података. Методолошки процес за оптимизацију ефеката протока података чине: идентификација процеса протока података, структурна декомпозиција процеса протока података, идентификација индикатора за праћење ефеката протока, јединице мере индикатора ефеката протока, идентификација ограничења индикатора ефеката протока података, идентификација ризика ефеката протока података и утврђивање акционих мера за митигацију ризика ефеката протока података. Елементи од посебног значаја за предложену методологију за оптимизацију ефеката протока података су: итеративни софтверски процес, структурна системска анализа, ефекти протока података, имплементација надгледања система, хибридни приступ у анализи ризика у фази одржавања софтвера и CASE алати.

Посматрајући првенствено трансакционе информационе системе предложена је нова методологија оптимизације ефеката протока података са интеграцијом ССА и анализе ризика у облику хибридне модификоване FMEA методе. Предложена методологија подразумева итеративни и инкрементални модел софтверског процеса као основу за успешну примену у пракси. Модификована хибридна FMEA метода која се посматрано у контексту софтверског процеса доменског софтвера, налази позиционирана у фази одржавања, као свој резултат даје списак мера које се примењују као активности модификације и наставка развоја софтверског решења. Неке од тако дефинисаних активности налазе се у фази дизајна, неке у фази имплементације дизајна, неке могу да се простиру на више фаза активности, али у сваком случају налазе се у ранијим фазама софтверског процеса, следећег циклуса. С друге стране, као улаз и предуслов за реализацију овакве предложене методологије, користе се елементи, посебно хијерархија декомпонованих процеса, спроведене ССА из ранијих фаза софтверског процеа из преходних итерација. Ова дуална међузависност, започиње условом иницијално спроведене структурне системске анализе.

У истраживању је дефинисано да доносиоци одлука у току спровођења анализе грешака уместо коришћења уобичајених нумеричких скала користе дефинисане лингвистичке исказе који су моделирани интервалним тип 2 трапезоидним фази бројевима (IT2TrFN), сматрајући да фази скупови омогућавају да се неизвесности и непрецизности квантитативно опишу на одговарајући начин. У овом раду дефинисано је да се релативна важност РФ-а задаје помоћу фази матрице парова упоређења са интервалним тип-2 трапезоидним фази бројевима, за разлику од конвенционалне методе где фактори ризика имају једнаку важност. Одређивање приоритета идентификованих грешака респектујући факторе ризика као и њихове тежине постављено је као задатак више-критеријумске оптимизације (ВКО) а ранг идентификованих грешака добија се применом конвенционалне COPRAS методе.

У експерименталном делу истраживања примењена је пројектована методологија за оптимизацију ефеката протока података на процесе у студентској служби Машинског факултета.

За сваку идентификовану грешку процењивана су три фактора ризика (РФ-а): озбиљност последице која настаје услед реализације грешке (С) - Severity, вероватноћа настајања грешке (О) - Occurrence, и могућност откривања грешке (Д) - Detection. Затим је утврђивана релативна важности РФ-а, у матричном облику, а помоћу лингвистичких термина који су моделирани интервалним тип 2 трапезоидним фази бројевима (IT2TrFN). Након одређивања важности тежинских вектора, извршена је провера конзистентности решења. Ранг идентификованих грешака софтвера одређиван је применом конвенционалне COPRAS методе. У овом истраживању, а на основу идентификованих узрока грешака из спроведене FMEA, препознате су две врсте мера које се могу предузети према нивоу припадности посматраном систему. Прва и основна врста су мере на модификацијама, дорадама и исправкама софтвера, како на нивоу логичког и физичког дизајна, тако и на нивоу имплементације - примењених технологија, хардвера и слично. Друго су мере на дефинисању и изменама процедура и реалних процеса који утичу на процесе информационог система. Када су у питању мере које отклањају пропусте у софтверу, дефинисане су мере које спречавају

настанак услова за испољавање грешака, мере које контролишу и спречавају сам догађај грешке и мере које ублажавају последице грешака. За оцену ефеката предузетих мера праћене су вредности дефинисаних индикатора пре и након примене корективних мера.

Резултат реализоване методологије показује да је побољшана ефикасност свих праћених процеса информационог система са аспекта протока података, или у 48 од 52 мерена индикатора, што је 92%. У току спроведеног експерименталног истраживања, утврђен је значај дефинисаног редоследа решавања узрока грешака. Међусобна условљеност промена у елементима софтверског решења значајно утиче на укупан резултат модификације и дораде у зависности од редоследа активности.

Кључне речи: структурна системска анализа, фмеа, ризик, проток података, развој софтвера, фази

Научна област: Машинство

Ужа научна област: Машинство и информационе технологије

ABSTRACT

In the doctoral dissertation, the problem of optimizing the effects of data flow in information systems and the development of a methodology for efficient management of data flow processes was researched. The methodological process for optimizing the effects of data flow consists of: identification of data flow processes, structural decomposition of data flow processes, identification of indicators for monitoring flow effects, units of measure of flow effect indicators, identification of data flow effect limiters, identification of data flow risk effects and determination of action measures for risk mitigation of data flow effects. Elements of special importance for the proposed methodology for optimizing the effects of data flow are: iterative software process, structural system analysis, data flow effects, implementation of system monitoring, hybrid approach in risk analysis in the software maintenance phase and CASE tools.

Observing primarily transactional information systems, a new methodology for optimizing the effects of data flow with the integration of SSA and risk analysis in the form of a hybrid modified FMEA method has been proposed. The proposed methodology implies an iterative and incremental model of the software process as a basis for successful application in practice. The modified hybrid FMEA method, which observed in the context of the software process of domain software, is positioned in the maintenance phase, as its result provides a list of measures that are applied as modification activities and extension of the development of the software solution. Some of the activities thus defined are in the design phase, some in the design implementation phase, some may extend to more phases of activity, but in any case they are in the earlier stages of the software process, of the next cycle. On the other hand, as an input and a precondition for the realization of such a proposed methodology, elements are used, especially the hierarchy of decomposed processes, implemented SSA from earlier phases of the software process from the previous iterations. This dual interdependence begins with the condition of the initially conducted structural system analysis.

In the research is defines that decision makers while conducting an error analyses instead of using the usual numeric scales use defined linguistic statements

modeled by interval type 2 trapezoidal fuzzy numbers (IT2TrFN), believing that phase sets allow uncertainties and inaccuracies to be described quantitatively. In this paper, it is defined that the relative importance of RF is given by the phase matrix of pairs of comparison with interval type-2 trapezoidal fuzzy numbers, in contrast to the conventional method where risk factors have equal importance. Prioritizing identified errors by respecting risk factors as well as their severity was set as a task of multi-criteria optimization (VKO) and the rank of identified errors was obtained by applying the conventional COPRAS method.

In the experimental part of the research, a designed methodology was applied to optimize the effects of data flow on processes in the Student service of the Faculty of Mechanical Engineering.

For each identified error, three risk factors (RF) were assessed: the severity of the consequence resulting from the realization of the error (S) - Severity, the probability of error (O) - Occurrence, and the possibility of detecting the error (D) - Detection. The relative importance of RF was then determined, in matrix form, using linguistic terms modeled by interval type 2 trapezoidal fuzzy numbers (IT2TrFN). After determining the importance of weight vectors, the consistency of the solution was checked. The rank of identified software errors was determined using the conventional COPRAS method. In this research, and based on the identified causes of errors from the conducted FMEA, two types of measures were identified that can be taken according to the level of belonging to the observed system. The first and basic type are measures on modifications, modifications and corrections of software, both at the level of logical and physical design, and at the level of implementation - applied technologies, hardware and similar. The second are measures to define and change procedures and real processes that affect the processes of the information system. When it comes to measures that eliminate deficiencies in the software, measures that prevent the occurrence of conditions for the occurrence of errors, measures that control and prevent the occurrence of the error itself and measures that mitigate the consequences of errors are defined. To assess the effects of the measures taken, the values of the defined indicators before and after the application of corrective measures were monitored.

The result of the implemented methodology shows that the efficiency of all monitored information system processes from the aspect of data flow has been improved, or in 48 out of 52 measured indicators, which is 92%. During the conducted experimental research, the importance of the defined order of solving the causes of errors was determined. The interdependence of changes in the elements of the software solution significantly affects the overall result of modification and refinement depending on the order of activities

Key words: structural systems analysis, fmea, risk, data flow, software development, fuzzy

Scientific field: Mechanical engineering

Scientific subfield: Mechanical engineering and information technology

САДРЖАЈ

СКРАЋЕНИЦЕ

ИС - Информациони систем

ИТ - Информационе Технологије

FMEA - Failure mode and effect analysis

SSA - Structured System Analysis

SDLC - Software development life cycle

RUP - Rational Unified Process

UML - Unified Modelling Language

RAD - Rapid Application Development

CASE - Computer Aided Software Engineering

ДТП - Дијаграм тока података

DBMS - Database management system

ВКО - Вишекритеријумска оптимизација

COPRAS - Complex Proportional Assessment

PIS - Позитивно идеално решење

NIS - Негативно идеално решење

TFN - троугаони фази број

IT2TrFN -Интервални тип 2 трапезоидни фази број

РФ - Фактор ризика (енглески: Risk factor)

GUI - Graphical User Interface (Графички Кориснички Интерфејс)

Saas - Software as a Service (Софтвер као Сервис)

СПИСАК ТАБЕЛА

СПИСАК СЛИКА

1. УВОД

Развој нових технологија, првенствено информационих технологија значајно утиче на реализацију пословних процеса, а самим тим и на ефективност пословања било које организације. Информациони системи (ИС) нас повезују са изворима података и процесима који те податке претварају у информације. Приступ великом скупу информација омогућава ефикасност и флексибилност управљања организацијом. Степен успешности пословања многих организација умногоме зависи од брзине, обима и квалититета протока података као и ефеката протока података. У многим организацијама информационо-комуникационе технологије су већ постале основно средство за рад. Коришћење информационих технологија омогућава ефикасно управљање подацима, али услед пораста обима података, масовне дигитализације, разноликих формата, дистрибуираности складишта података и сличних разлога, долази до отежаног проналажења праве информације у банкама/складиштима података и успоравања приступа траженим подацима. Стога се акценат помера са проблематике ефикасног чувања података, на брзу трансформацију података и приступ информацији у правом тренутку.

На примену информационих технологија утичу многобројни различити фактори ризика, а последице су по правилу значајне за остваривање циљева организације. У већини организација не постоје организовани и ефикасни процеси управљања ризицима информационих система. Препрека бољем управљању овим ризицима у организацијама је одсуство дефинисаних методологија и процеса управљања ризицима информационих система, софтвера, хардвера, мрежа и људи. Доношење оквира за управљање ризицима на методолошки начин, резултирало би смањењем губитака у организацијама. Установљавање процеса управљања ризицима софтвера и шире гледано ризицима информационих система, подразумева потребу за ресурсима и оспособљеним кадровима. Неадекватан третман ових ризика може изазвати неке од следећих проблема: пословање са губицима због слабо реализованих софтверских пројеката, недоступност података и информација, нарушавање интегритета пословних процеса, губитак података и интелектуалне својине и слично. Да би се ови

негативни ефекти у пословним процесима смањили, потребно је системски и организовано управљати ризицима. У овом тренутку не постоји јединствен прихваћен и стандардан алгоритам по коме се врши процена ризика информационих система.

У том смислу, предмет разматрања ове докторске дисертације јесте разматрање процеса протока података и истраживање методологије за ефикасно управљање процесима протока података. С обзиром да процеси протока податка имају директан и индиректан утицај на ефикасност посматраног информационог система, односно последично на ефикасности пословања организације, ради свеобухватног разматрања утицаја протока података надаље се користи термин ефекти протока податка, под којим се подразумевају директни, посредни и непосредни утицаји на ИС од стране атрибута (елементи којима се описују својства) протока података.

Ризици који се могу идентификовати код процеса протока података у ИС, су потенцијално изражени губици у мерним јединицама ефекта протока података, чијом митигацијом се превазилазе предвидиви губици и омогућава управљивост процеса протока података. Цео поступак истраживања ефеката протока података: идентификација процеса протока података, структурна декомпозиција процеса протока података, идентификација индикатора за праћење ефеката протока, јединице мере индикатора ефеката протока, идентификација ограничења индикатора ефеката протока података, идентификација ризика ефеката протока података и утврђивање акционих мера за митигацију ризика ефеката протока података чини методолошки процес који се у истраживању назива методологија за оптимизацију ефеката протока података.

Основни научни циљ ове докторске дисертације је показати да се савременим концептом на бази интеграције структурне системске анализе и анализе ризика, може унапредити управљање ефектима протока података у информационим системима, а са посесним осврtом на софтвер као кључну компоненту информационог система.

Основна хипотеза дефинисана је на следећи начин:

- интеграцијом структурне системске анализе (SSA - Structured System Analysis) - методе за анализу и функционалну декомпозицију информационог система и FMEA (Failure Mode and Efects Analysis) - методе за идентификацију, анализу и оцену ризика, могу се: идентификовати процеси протока података, извршити структурна декомпозиција процеса протока података, идентификовати индикатори за праћење ефеката протока, одредити јединице мере индикатора ефеката протока, идентификовати ограничења индикатора ефеката протока података и идентификовати ризици ефеката протока података за посматрани информациони систем.

Додатне хипотезе истраживања су:

- интеграцијом SSA и FMEA пројектује се методологија за управљање ефектима протока података у информационим системима,

- оптимизација ефеката протока података у информационим системима реализује се применом пројектоване методологије за управљање ефектима протока података базиране на интеграцији SSA и FMEA.

У практичном делу истраживања биће примењена SSA метода за анализу и функционалну декомпозицију информационог система службе за студентске послове Машинског факултета (у даљем тексту: студентска служба). На декомпоноване процесе у оквиру ИС студентске службе биће примењена FMEA метода за оцену ризика. FMEA метода такође ће омогућити да се изврши идентификација грешака процеса и изврши њихова приоризација. На основу одређеног приоритета грешака, одредиће се приоритет мера које треба да се реализују у циљу елиминисања истих грешака и на тај начин реализује поступак оптимизације ефеката протока података.

У раду ће бити примењене квантитативне методе за анализу посматраних фактора којима се описују ефекти протока података, методе вишекритеријумске анализе за рангирање идентификованих грешака процеса протока података,

квалитативне методе за оцењивање фактора ризика идентификованих грешака, и примењене методе софтверског инжењерства за пројектовање софтверских решења.

Друштвени допринос односи се на нову методологију за управљање ефектима протока података у информационим системима за коју се очекује да ће наћи ширу примену (у неком пословном окружењу) за моделирање и решавање проблема управљања и одлучивања, посебно оних који се односе на информационе системе оријентисане на обраду података и процес развоја и одржавања софтверских решења.

Нова методологија за управљање ефектима протока података, могла би се прилагођавати зависно од контекста, тј. специфичне ситуације, у различитим информационим системима.

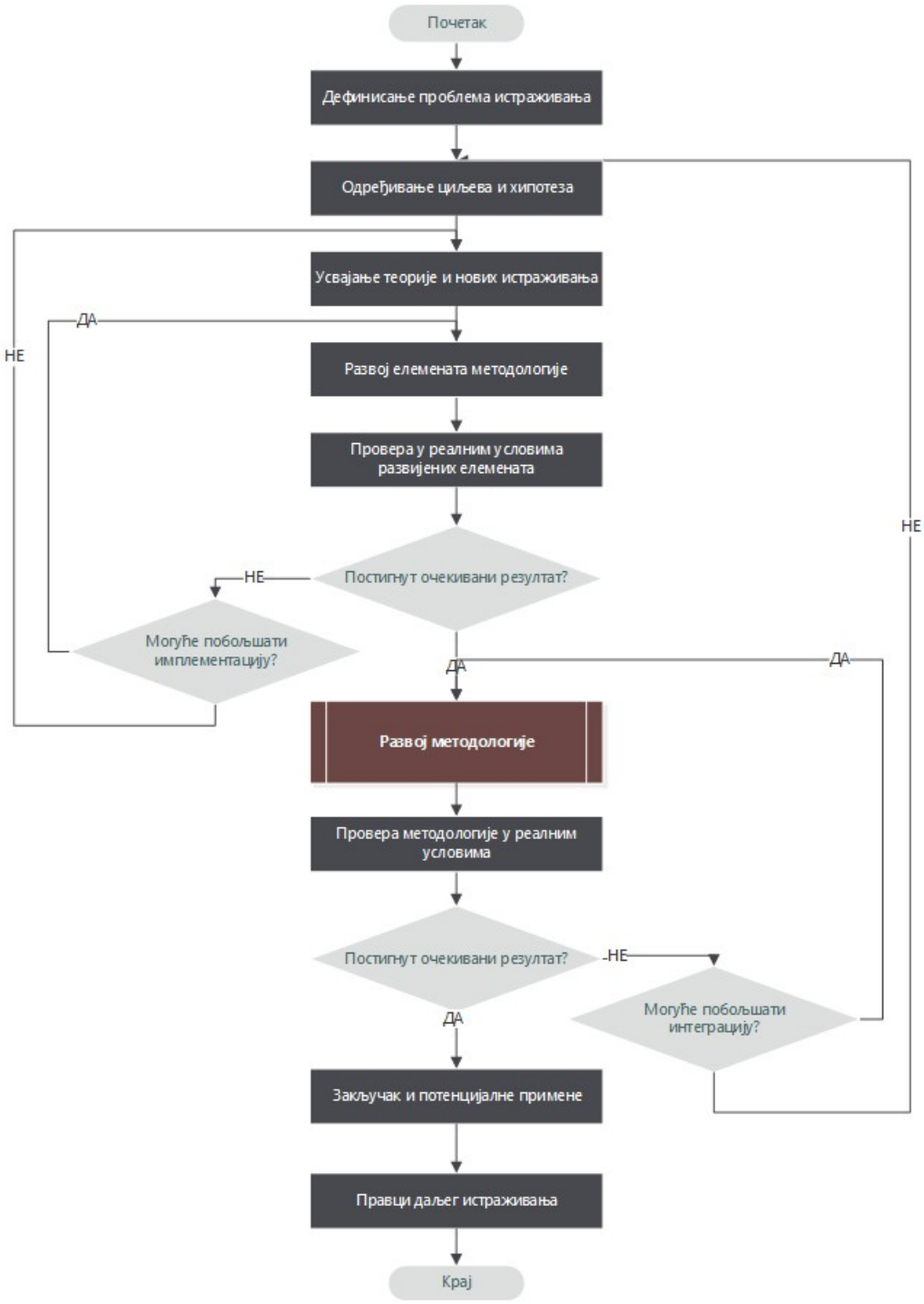
План истраживања одвијаће се кроз следеће фазе:

У првој фази рада планира се систематизација сазнања у области примене CCA и FMEA, и критичка анализа одговарајуће литературе.

У другој фази планира се развој нове методологија за управљање ефектима протока података на бази интеграције структурне системске анализе (SSA) и анализе ризика (FMEA метода).

У трећој фази планира се експериментална провера успешности развијене методологије на примеру информационог система студентске службе Машинског факултета.

У последњој фази истраживања, фокус ће бити на анализи истраживања и дискусији резултата на основу којих ће бити изведени закључци. Алгоритам планираног истраживања приказан је на слици 1.



Слика 1: Алгоритам истраживања

2. ТЕОРИЈСКА ОСНОВА ЗА РАЗМАТРАЊЕ ПРОЦЕСА ЕФЕКАТА ПРОТОКА У ИС

2.1. Информациони системи

Појам информациони систем Лукас дефинише као скуп организованих процедура, које када се спроведу, обезбеђују информације за подршку организацији. [1]. Информациони системи су данас широко присутни у свим сферама живота. Користе се како у свакодневном животу појединаца, тако и у различитим организацијама, пословним системима, научним истраживањима и слично. Скоро сваки производ данас има неку врсту уграђеног рачунара, и скоро свака услуга која се пружи на тржишту има везе са неким информационим системом. У овом раду, пажња је усмерена на ефекте протока података у оквиру рачунарских информационих система (у даљем тексту користићемо израз информациони систем, мислећи заправо на рачунарски информациони систем).

Први рачунарски информациони системи служили су поједностављено гледано за "аутоматску обраду података" (АОП). Развојем информационих система, појављују се различите напредне врсте система као што су: управљачки информациони системи (Management Information Systems - MIS), системи за подршку одлучивању (Decision Support Systems - DSS), експертни системи као поље примене вештачке интелигенције (Expert Systems - ES) и разни други. Информациони системи за обраду података, познати и као датацентрични информациони системи, имају за нас посебан значај. Било какво управљање организацијом подразумева доношење одлука. Да би се доносиле одлуке и решавала постављена питања, потребно је имати адекватне информације. Такве информације обезбеђују пословни информациони системи и пословни софтвери који су део ИС-а.

Према Гартнер-у, вредност светског тржишта пословног софтвера за 2017. годину је 355 милијарди долара, а у 2018. години 389 милијарди долара, што је повећање за 9,5%. За 2019. годину, предвиђање је да ће се вредност попети на 421 милијарду долара. Укупна процењена вредност роба и услуга из области информационих технологија за 2019. годину је 3,784 трилиона долара [2].

Информациони систем је увек једна целина, таква да представља скуп компоненти које су у складу и извршавају одређене активности. Као и сваки систем подразумева скуп елемената који су међусобно повезани. Рачунарски информациони систем у општем случају садржи ресурсе које можемо сврстати у следеће категорије: хардвер, мреже, софтвер, подаци и људи. Информациони систем прикупља, обрађује, организује и меморише податке. Подаци се складиште у складиштима података (базама података), а систем обрађене податке саопштава крајњим корисницима система. То могу људи али и други информациони системи. Хардвер као ресурс представља све врсте рачунарских уређаја. Софтвер обухвата све врсте програма, апликативне и оперативне, софтерске библотеке, разне софтверске контролне процедуре и слично. Подаци као есенцијални ресурс се налчешће јавља у облику неке базе података или базе знања. Мрежа подразумева комуникационе уређаје, медије и протоколе. А људски ресурс се може поделити у две групе, стручњаке који развијају и администрирају систем и кориснике система. Суштина сваког информационог система је трансформација података. Она се обавља кроз разне процесе обраде. Резултат обрада су разне форме информација за групе крајњих корисника. Обрада података у систему може се састојати од различитих активности као што су улаз података, обрада података, меморисање података и излаз података.

2.2. Софтверски процес

Последњих деценија наступио је веома убрзани развој рачунарства, тако да су се развиле различите али међусобно повезане дисциплине: софтверско инжењерство, рачунарство, програмирање, информатика, рачунарске науке, информационе технологије, итд. Софтверско инжењерство као посебна дисциплина, којом се баве софтерски инжењери, односи се на организовани и систематичан развој софтвера као производа за тржиште. Развој који треба да применом дефинисаних правила и потврђених искустава из професионалне праксе, доведе до стварања поузданог, квалитетног и ефикасног софтверског производа. Софтверски производ је производ намењен купцу, односно кориснику. Садржи скуп рачунарских програма и припадајућих елемената потребних за коришћење. Оно што ипак чини значајну разлику у односу на друге инжењерске области, је чињеница да софтвер иако производ, није материјалан, није нешто што се може физички додирнути. Ако ово изузмемо, као у свакој инжењерској дисиплини, кључни појмови су: процеси, стандарди, правила, методе, тимски рад, индивидуалне вештине, пракса, квалитет и слично.

Софтверски производ који се развија може бити предвиђен за одређеног корисника, и тада је прилагођен тачним потребама тог корисника (енглески: customized software), а може бити предвиђен за тржиште и тада обично садржи функционалности које одговарају широком скуп потенцијалних корисника (енглески: generic software product). Уобичајено се за софтверски производ користи скраћени израз: софтвер, или такође раширено: софтверски систем.

За модеран софтвер се подразумева да мора бити квалитетан. То значи да обавезно мора испуњавати неколико основних услова. Најпре, софтвер мора бити употребљив, треба да садржи функционалности које корисници очекују од њега. Даље, осим што ради оно за шта је намењен, мора имати и високе перформансе. Осим ефикасности мора имати висок степен поузданости и мора радити на очекивани начин. На крају, да би се обезбедило функционисање у условима сталних промена у окружењу, софтвер мора имати могућност одржавања и измена у току експлоатације. Да би се производио такав софтвер неопходно је

примењивати методе које су део софтверског инжењерства. Софтверско инжењерство је не само стручна, већ и научна дисциплина. Бави се пре свега моделима и методама производње софтвера.

Софтверско инжењерство се почело интензивно развијати почетком 70-тих година 20-тог века. Велики напредак на пољу хардвера и појава рачунара треће генерације довели су до очекивања сложенијих и комплекснијих софтвера, како на пољу апликативних програма тако и на пољу оперативних система, на пример мулти-таскинг системи. Многи тадашњи софтверски пројекти завршени су неуспешно, најчешће због прекорачења рокова, премашивања буџета или уопштено због лошег управљања ресурсима. Позната као "софтверска криза" ова појава указала је на недостатке тадашњих, у суштини неформалних техника развоја софтвера. Приступ где је програмирање схватано као индивидуална вештина са дозом уметности, није могао да се успешно примени на велике софтверске пројекте.

Књига „Уметност рачунарског програмирања - 1.део: Основни алгоритми“ ("The Art of Computer Programming - Volume 1: Fundamental Algorithms) која је објављена је 1968. године, а коју је написао Доналд Ервин Кнут, један од најпознатијих програмера у историји, показује суштинску везу између математике и програмирања. Да не буде забуне, Кнут у књизи не говори о развоју софтвера већ о алгоритмима.

Као одговор на "софтверску кризу" почиње развој софтверског инжењерства и сложенијих метода развоја софтвера. Програмирање постаје и предмет интересовања академских кругова. Да би се управљало пројектима са великим бројем програмера и да би се обезбедила контрола комплексних пројеката преузете су многе методе из "старијих" инжењерских техничких струка.

Како је то приметио Dijkstra [3], системи постају све већи и бржи и као резултат долази до много више простора да дође до грешака. John von Neumann је разматрао проблем изградње поузданог компјутера из непоузданих компоненти, али је студија престала после von Neumann-а и када су транзистори заменили непоуздане електронске цеви. Сада смо суочени са овим проблемом на вишем нивоу и то није проблем управљања, већ научни изазов.

Неки софтверски производи могу се сматрати најкомплекснијим системима које је направио човек. Развој тако сложених система подразумева употребу техника и метода које успешно могу да се примене и на сложене софтверске пројекте, а да обезбеде перформансе и поузданост. Развој једне такве методологије која користи како аналитичке, тако и дескриптивне алате, са циљем повећања ефикасности путем смањења непоузданости софтвера, развијена је у оквиру ове дисертације. Производња софтвера захваљујући напретку софтверског инжењерства, сада је активност која има већу дозу предвидљивости. Софтверско инжењерство се и даље развија, могућа су даља унапређења и нови приступи.

2.2.1. Процес развоја софтвера

Софтверско инжењерсво покрива комплетан животни циклус софтвера (енглески: software development life cycle - SDLC). Процес развоја софтвера, или како се то најчешће каже, софтверски процес, представља скуп активности на производњи софтвера које се спроводе на један систематичан, контролисан и ефикасан начин користећи формалне методе за спецификацију, анализу, дизајн, имплементацију и одржавање софтвера.

Неке опште прихваћене парадигме у развоју софтверских система јесу: моделовање, итеративност, документовање, архитектура и слично. То су познати принципи који се осим при развоју софтвера примењују и у другим различитим подручјима рада.

- Моделовање, избор модела, одређивање нивоа детаљности модела, везу између модела, веза модела према другим нивоима апстракције;

- Итеративност, поновљање активности ради усклађивања, дотеривања, побољшања софтверског производа;

- Документовање, опис софтверске архитектуре, функционалности, интерфејса и слично, путем скупа различитих докумената, табела, кодова, дијаграма и слично.

- Архитектура, декомпоновање сложене структуре на његове елементе. Користи се у свим фазама, а посебно у анализи и дизајну. Омогућава бољу разумљивост сложених система.

Софтверски процес је скуп активности подељених у фазе које се реализују током развоја софтверског производа. Најпознатији модел процеса, секвенцијални модел, као први широко примењиван, садржи фазе које се у неком облику и редоследу појављују у већини модела: Спецификација захтева, Анализа, Дизајн, Имплементација, Тестирање, Испорука и одржавање [4].

У фази израде спецификације и у току анализе, утврђује се шта софтвер треба да ради, шта треба да омогући корисницима. Анализирају се захтеви корисника. Захтеве суштински можемо поделити на две врсте:

• Функционални захтеви. Описују функције будућег система. Одређују како систем треба да се понаша у разним сценаријима и ситуацијама. Описују шта тачно треба да омогући, какве резултате и излазе треба да изда за одређене улазе.

• Нефункционални захтеви. Представљају ограничења и услове у којима ће систем радити. То може бити захтевана брзина рада, тачност, поштовање неког стандарда, врста оперативног система, и слично.

Током анализе захтева састављају се модели система, како постојећег тако и будућег. Модел представља апстракцију система. Самим тим, део детаља се занемарује, посматра се поједностављена слику система. Систем се посматра из различитих перспектива, одакле следи могућност формирања различитих модела. Ако су модели представљени помоћу дијаграма, тада се повећава разумљивост коисницима, а осим тога такви графички модели постају прецизнији и информативнији од описа направљених помоћу природног језика. Најважније две перспективе из којих треба посматрати софтверски систем су понашање и структура система. Понашање система треба да обухвати процесе који се одвијају у оквиру система, трансформације података, реакције на одређене догађаје и слично. Структура система обухвата архитектуру система и подсистема, као и структуру података који су део система. Често је потребно посматрати и окружење у којем систем ради, контекст и одредити које су границе система.

Модел тока података посматра систем из перспективе понашања и описује процес обраде података. Систем се моделира као скуп процеса обраде података, токова података између тих процеса, и складишта података која могу бити извор или одредиште. Моделирање тока података постало је прихваћена и популарна метода после објављивања књиге: T. de Marco: “Structured Analysis and System Specification” 1978. И даље се доста користи у оквиру разних функционално оријентисаних приступа развоју софтвера Представља важан део методологије која је предложена у овој дисертацији.

Објектни модели у својој основи, имају идеју спајања у једну целину перспективе структуре и понашања. Објектни модели су новији приступ, логично користе се у оквиру објектно оријентисаних метода за развој софтвера. Booch, Rumbaugh и Jacobson, спајањем својих оригиналних метода и приступа, установили су јединствену методу развоја софтвера: Rational Unified Process (RUP), и успоставили су стандардну нотација за цртање дијаграма: Unified Modelling Language (UML). У оквиру објектних модела, софтверски систем се дефинише помоћу класа објеката. Класе објеката и односи између тих класа су суштина ових модела. Како се објекат састоји од атрибута и операција, објектни модели следећи своју основну идеју комбинују особине модела тока података и модела ентитета и веза.

У току дизајнирања система одређује се који су то саставни делови система, на који начин ће систем радити и како ће међусобно комуницирати компоненте система. Ако се у фази анализе одређује шта ће систем радити, у фази дизајна одређује се како ће систем радити. Суштина фазе дизајна система је потрага за алтернативним решењима са циљем испуњења дефинисаних захтева. Резултат дизајна система треба да буде довољно прецизни опис структуре система, опис компоненти, подсистема, међусобних интерфејса, корисничког интерфејса система, структура података, па и алгоритама који ће се користити. Дизајн је високо креативан и итеративни поступак.

Када се дође до фазе имплементације, прелази се на конкретно програмирање, уз коришћење конкретних програмских језика, алата и сложенијих пакета и радних оквира. Последње итерације дизајна обично су довољно детаљне

да омогућавају паралелно започињане имплементације. Исто тако, упоредо са имплементацијом могуће је обављати и активности тестирања софтвера. Имплементација новог софтвера је претварање пројектантских решења вишег логичког нивоа у конкретан програмски код и елементе базе података. Осим израде новог кода, потребно је обавити имплементацију пројектоване технологије и система у целини. Имплементација се пажљиво планира. Многи аутори сматрају да се целокупан план може поделити на четири основне целине:

(1) Програмирање;

(2) Инсталација и тестирање;

(3) Обука корисника и

(4) Увођење система.

Обично се по изради софтвера приступа некој врсти провере реализованог система, верификацији да ли је све испуњено по захтевима и валидацији да ли систем ради исправно. Одржавање наступа као и у другим инжењерским дисциплинама после увођења система у употребу. Најчешће се и после увођења у употребу, софтвер и даље поправља, дограђује и модификује на основу промена окружења и појаве нових захтева. Процес мењања софтвера називамо одржавање односно еволуција. Мада неки аутори под одржавањем подразумевају планиране активности које се састоји од мањих промена, а за еволуцију сматрају да је шири појам и да је то процес који доводи до већих и позитивних промена у архитектури софтвера, у овом раду су то практично синоними. Могу се препознати следећи типови измена софтвера:

• Одржавање софтвера. Измене се раде због нових захтева. Мења се функционалност без крупних измена структуре софтвера.

• Архитектурна трансформација. Значајна промена архитектуре софтверског система са могућом променом функционалности. Пример може бити промена са класичне десктоп апликације која ради у локалној мрежи на Web апликацију која ради у оквиру Интернет прегледача и ослања се на Web сервисе.

• Софтверско реинжењерство. Нема нових функционалности, нити се значајније мења архитектура. Систем се мења у модернији облик ради лакшег одржавања, документовања, повезивања и слично. Прелаз са једног застарелог на други новији програмски језик.

Такође, може се разликовати да ли се активности у измени софтвера у оквиру одржавања, обављају да би се исправиле уочене грешке - корективно одржавање, или ради прилагођавања софтвера новом контексту - адаптацијско одржавање. Обично се у свакој новој верзији софтвера комбинују оба принципа.

Савршено дизајниран, развијен и тестиран софтвер у идеалним условима експлоатације без измена захтева, не захтева одржавање. У пракси међутим, показује се велика потреба за модификацијама софтвера. Трошкови одржавања расту са старошћу апликација. Некада се тешко може одредити граница између одржавања и развоја постојећег система. У предложеној методологији у овом раду, процес одржавања се ослања на исте активности које се користе у фази анализе, дизајна и имплементације система. Потенцира се важност одржавања система за његов животни циклус развоја.

2.2.2. Модели софтверског процеса

Развој софтверских система подразумева велики број активности, које се фазно реализују. У зависности од приступа и начина размишљања оних који развијају систем, као и од особина и врсте система за који се софтвер развија, могуће је појединачне активности организовати на различите начине. Модел развоја је апстракција процеса развоја. Сваки постојећи модел представља процес развоја на свој посебан и специфичан начин. Модели развоја се међусобно разликују, а посебно је важно колики значај посвећују одређеним фазама и активностима у току развоја. Модели се најлакше препознају по свом логичком редоследу реализације појединих фаза и активности. Поједине активности међусобно су често вишеструко условљене на сложен начин. Резултати активности употребљавају се у другим активностима,. Модел развоја је некада условљен природом пројекта. У принципу, који ће се модел развоја користити, као и које методе и алати за развој, обично зависи од особа које учествују у развоју. Модели развоја обезбеђују шаблоне развоја софтвера. Они су ту да обезбеде систем и логички оквир за планирање, организовање и управљање разним активностима у току развоја софтверских система. Неки од познатих модела развоја софтвера су :

- Модел водопада,
- Итеративни модел,
- Модел рапидног развоја апликација,
- Модел прототипског развоја,
- Спирални модел,
- Модел заснован на компонентама,
- Модел унифицираног процеса развоја
- Модели агилног развоја
- "Синхронизуј и стабилизуј" модел

- "Изгради и фиксирај" модел

- Комбиновани модели

2.2.2.1. Модел водопада

Модел водопада (енгл. waterfall) је најстарији широко примењен модел развоја софтвера. Први формални опис модела водопада dat је у [5]. Софтверски процес је грађен као низ временски одвојених активности. Модел је добио такав назив зато што је у њему ток активности такав да се постигнути резултат преноси из једне фазе у другу па визуелна представа подсећа на водопад. Овај модел развоја подразумева систематичан приступ активностима. Активности се реализују по строго дефинисаном редоследу. Познат је и као секвенцијални модел развоја. Свака фаза има свој дефинисани почетак, крај и очекиване резултате без којих не почиње наредна фаза. Овај модел је убедљиво највише примењиван до данас и представља основу за многе друге моделе развоја. Често је критикован али је и даље у широкој употреби као ефикасан за структурирање и управљање малим пројектима.

Основне карактеристике секвенцијалног процеса развоја су према [6]:

1. кроз фазе се пролази редом, када се фаза једном заврши више нема повратка на њу;

2. на следећу фазу се прелази тек пошто текућа буде комплетно завршена;

Овај модел је историјски први модел процеса развоја софтвера. Настао је по узору на производне и грађевинске пројекте [6]. Код оваквих пројеката, није ефикасно враћање на претходне фазе. На пример, једном завршен део грађевине не поправља се без велике нужде, алати и ресурси из претходних фаза се ослобађају. Софтверски пројекти се значајно разликују од класичних индустријских или грађевинских врста пројеката, софтверски производ треба развијати са инжењерским приступом, али софтвер није опипљив. Развој

софтверског производа тако пролази кроз низ корака. Излаз из сваке фазе развоја треба бити документован и верификован.

Модел водопада или секвенцијални модел развоја софтвера подразумева само један пролаз кроз следеће фазе развоја софтвера према [4]:

1. Спецификација захтева

Спецификација захтева, подразумева јасно дефинисање шта одређени софтвер треба да ради и које карактеристике треба да има у складу са функционалним и нефункционалним захтевима. Софтвер увек представља само део неког система. Потребно је идентификовати који су захтеви усмерени према софтверу. Треба одредити које циљеве треба постићи развојем софтвера.

2. Анализа

Анализа, представља сагледавање елемената захтева и формирање концептуалног модела који представља грубу скицу и основни логички модел софтверског система [7].

3. Дизајн

Дизајн софтверског производа, некада се користити и израз пројектовање, је фаза развоја у току које се дефинише архитектура система. Обично обухвата дизајн корисничког интерфејса, маски за унос података, излазних форми, база података, процедура за обраду података и слично. Дефинише се структура и интеракција између компоненти софтверског система. Ако се користи нека објектно оријентисана методологија софтверски систем се описује помоћу скупа UML дијаграма као што су: дијаграм класа, дијаграм секвенци, колаборациони дијаграм, дијаграм стања, итд.

4. Имплементација

Имплементација, подразумева писање програмског кода на основу модела софтверског система креираног у фази дизајна. Резултат ове фазе је извршна везија софтвера. Обично обухвата и обуку корисника. Што је пројектовање урађено детаљније, то је програмирање једноставније. Ако се систем састоји од

више засебно развијених компоненти, потребно је обавити интегрисање компоненти у функционалну целину.

5. Тестирање

Тестирањем се проверавају све програмске целине. Тестирање може да открије алгоритамске грешке , семантичке, као и синтаксне мада се оне сада по правилу детектују раније у фази програмирања. Тестирање треба да обухвати и проверу да ли систем испуњава постављене захтеве и да ли софтвер исправно ради у различитим сценаријима и посебно граничним случајевима.

6. Испорука и одржавање

Испорука и одржавање подразумевају предају софтвера кориснику на коришћење. Одржавање је најдужа фаза развоја, током које се производ константно унапређује. Промена окружења, нове технологије, нови функционални захтеви корисника и исправке грешака које нису уочене током тестирања, основни су разлози за иновирање софтверског производа у фази одржавања.

Предности модела водопада су: прецизно дефинисани процес, стандардизоване активности, верификација извршених активности и резултата, обавезна документација, итд. Највећи недостаци модела водопада су: нефлексибилност у распореду активности, иреверзибилност и одсуство повратне спреге између фаза, дуготрајан процес развоја, софтвер доступан на крају активности развоја, високи развојни трошкови. Основни проблем код овог модела је што није могуће у старту предвидети све захтеве и проблеме. У пракси се показало да део захтева постаје познат тек када крајњи корисник почне да користи софтвер, што је и битан елемент предложене методологије у овом раду.

Проблем који се јавља код примене овог модела развоја, је немогућност сагледавања свих потребних функционалности решења које се развија у раним фазама процеса као последица велике сложености савремених софтверских система. Модификовани модел водопада уводи могућност преклапања активности развоја и повратне спреге између фаза. Модификовани модел водопада

омогућавањем повратка на раније реализоване фазе развоја смањује опасност од преношења озбиљнијих грешака кроз све фазе развоја до самог краја.

2.2.2.2. Итеративни модел

Основа итеративног развоја софтвера је развој у више поновљених циклуса (итерација). У свакој итерацији се понављају све фазе развоја које постоје и код модела водопада [8]. У свакој итерацији се поправља и дорађује урађено, додају нове могућности и функционалности софтверског решења које се развија. Од плана развоја зависи колико ће се у конкретној итерацији радити на доради и усавршавању раније урађених делова система, а колико на изради нових компоненти система које се додају претходном решењу. После имплементације сваке нове верзије система, користе се повратне информације од корисника и на основу тога сагледава и планира се даљи развој.

Модел итеративног развоја је креиран тако да се лакше превазиђе проблем детаљног и поузданог дефинисања захтева на почетку развоја, као и чињеницу да није могуће предвидети све проблеме у раду пре почетка коришћења и да се нови захтеви често дефинишу тек када је систем у употреби. У зависности од сложености проблема развој се спроводи у више итерација. Свака итерација треба да има дефинисане очекиване резултате и да буде наставак рада на резултатима из претходне итерације. Итерације омогућавају ефективност, лакше сагледавање исправности процеса развоја. Природно је и познато да се људи лакше суочавају са сложеним пословима итеративним приступом где се после иницијалне фазе и стварања језгра решења изводе мала и константна унапређења.

У почетној итерацији се најпре развија неки иницијални скуп функционалности. Дефинисање основних функционалности софтвера омогућава да се већ прва верзија решења може применити у раду. Дизајн основне архитектуре система се обично реализује у првој итерацији. Нове функционалности које се додају тренутно развијеном производу представљају

инкремент претходног система па се овај начин развоја често назива и итеративно-инкрементални.

Неке од предности модела су:

- Видљиви резултати развоја,

- Смањени ризик неуспеха комплетног развоја,

- Брзо расположив функционални софтверски производ,

- Повратна спрега од стране корисника,

- Лакше планирање развоја и употреба ресурса.

2.2.2.3. Модел рапидног развоја апликација

Модел рапидног развоја апликација (енглески: Rapid Application Development - RAD) је инкрементални модел процеса развоја. Софтверски производ се развија у кратким развојним циклусима, обично у трајању од 2 до 3 месеца. Фазе развоја: анализа, дизајн, имплементација и тестирање су наглашено сабијене у кратке итеративне развојне циклусе. Производ се развија са малим тимовима. Карактеристике модела су: мали развојни тимови, развој са кратким итеративним развојним циклусима, директна и ефективна комуникација чланова развојног тима и јасна дефинисаност пројектних циљева. Такав приступ омогућава паралелни рад на моделу података, моделу процеса, па и израду прототипа који се одмах потом проверавају, па све поново у новој итерацији. У оваквом начину развоја максимално се користе CASE алати, генератори кода и слични напредни помоћни софтвери. Веома важно за овакав развој је активно укључивање и сарадња са свим стејкхолдерима. Тестирање које је саставни део сваке фазе развоја, а које на сваком инкременту спроводе чланови развојног тима и исто тако и сами корисници, обезбеђује да брзи развој буде и проверен и потврђен.

Рапидни развој апликација има за циљ знатно бржи развој и већи квалитет него у традиционалном моделу водопада. Кључни циљеви RAD концепта су

повећана брзина развоја уз висок квалитет излазног призвода и контролисани трошкови.

У оквиру овог концепта препознају се четири основне компоненте:

1. Методологија

Користе се формализоване најнапредније технике развоја, води се рачуна да се све активности спроведу исправно уз потпуно документовање;

2. Управљање

Пројектом се тако управља да се обезбеди брзи развој, за брзо дефинисање корисничких захтева користе се средства у форми практикума, а у свим фазама RAD развоја користе се технике временског управљања;

3. Људски фактор

Чланови развојног тима морају бити добро обучени, треба да буду потпуно упознати са методологијом, као и да напредно користе софтверске алате;

4. Алати

Аутоматизација развоја система постиже се употребом CASE алата, коришћењем софтвера за аутоматску проверу исправности база података, генерисање кода, израду прототипова и слично, све то са циљем ефикасног итеративног развојног поступка;

Најважнија предност RAD модела је повећана брзина развоја производа са методом прототипског развоја у основи, али брзина развоја не треба да постане сврха сама себи. Треба водити рачуна да брзина развоја не одведе развој ка импровизацији и изради само приручних и привремених решења. Потребно је развијати компоненте система са визијом каснијег повезивања и модуларизације.

2.2.2.4. Модел прототипског развоја

Модел прототипског развоја је итеративан модел. По овом моделу развоја најпре се развија почетни-иницијални модел софтверског производа. Овај почетни

модел софтвера обично садржи елементе корисничког интерфејса како би корисници који су укључени у развојни процес од самог почетка могли да искажу своје мишљење и захтеве. Дефинисана функционална спецификација будућег софтвера је у овом моделу развоја почетна основа за почетак развоја. Прототип се иницијално развија, тестира и касније по потреби дорађује. Прототип представља основу, шаблон за развој комплетног производа. Овакав приступ је практичан када се на почетку не познају довољно захтеви корисника и када се очекује поступно дефинисање комплетних захтева. Користи се и када је потребно извести симулацију рада производа да би будући корисник стекао увид у очекивани начин рада и могућности софтвера, као и када развојни тим жели проверити неки приступ или карактеристику система.

Најпре се спроводи прикупљање најважнијих захтева корисника. У сарадњи чланова развојног тима и самих корисника дефинишу се циљеви развоја производа. Спроводи се поједностављени дизајн са пажњом на елементе софтвера који директно комуницирају са корисником. Тако креирани прототип се модификује итеративно све док прототип не задовољи захтеве корисника. Прототип служи и дизајнеру као средство за боље разумевање захтева, и као средство комуникације са будућим корисницима. У суштини прототипски развој служи и као механизам за идентификовање корисничких захтева.

Једна од највећих предности примене модела прототипског развоја је то што је корисник директно ангажован на развоју, па може мењати своје захтеве и боље разумети како су захтеви интерпретирани од стране развојног тима, што унапређује квалитет целокупног процеса. Осим тога овако обезбеђене прототип радне верзије производа служе развојном тиму за проверу функционалности, адаптибилности, перформанси делова решења.

Овај модел има и своје недостатке. Могући су компромиси у имплементацији са циљем што брже изградe прототипа. Примењени алгоритми, структуре, технологије, решења, да би се демонстрирале поједине функционалности производа, су обично мањег квалитета од потребних као део коначног производа. Корисник може имати нереалну процену производа не схватајући да у реализацији прототипа нису разматрани аспекти квалитета,

одржавања и администрирања система. Значајно је да сви укључени актери разумеју да се прототип развија као средство за дефинисање захтева, за приказ изгледа будућег производа.

2.2.2.5. Спирални модел

Спирални модел представља комбинацију модела водопада и прототипског модела развоја. Користи се за велике и комплексне пројекте. Спирални модел [9] је представљен широј публици 1988. године.. Овај модел спаја добре особине модела водопада и модела прототипског развоја. Као специфичност садржи активности анализе ризика. Посебно је препоручен за развој скупих и важних пројеката.

Визуелна представа модела је у облику спирале. Модел предвиђа четири фазе развоја које се понављају у циклусима:

1. фаза - планирање

У овој фази кроз комуникацију чланова развојног тима и корисника реализује се дефинисање циљева, одређивање пројектних алтернатива и идентификовање ограничења процесу развоја,

2. фаза - анализа алтернатива и ризика

Ова фаза садржи активности анализе алтернатива и идентификовања ризика у управљању пројектом,

3. фаза - инжењеринг

Фаза инжењеринга је продуктивна фаза, у којој се ради на развоју нових елемената производа и на унапређењу раније реализованих делова система, тестирању, документовању и инсталирању решења код корисника,

4. фаза - процена корисника

Фаза процене реализованих резултата развоја софтверског производа.

Спирала развоја у овом моделу састоји се вишеструких узастопних циклуса развоја. Основна идеја модела је да се одређени редослед корака развоја и одржавања понавља. У почетнин циклусима ради се са вишим степеном апстракције а касније се прелази на детаље. Сваком итерацијом се прогресивно развијају комплетније и комплексније верзије софтверског производа. Број активности у фази инжењеринга се повећава како се циклуси удаљавају од центра спирале, од почетка развоја. Самим тим сваки нови циклус спирале доноси комплетнији производ, али и веће трошкове због обима и природе спроведених активности.

У првом-почетном циклусу спирале, слично моделу водопада најпре се прикупљају захтеви и планира развој. На основу прикупљених почетних захтева обавља се анализа ризика захтева. Ако се установи одређени ниво неизвесности у решавању захтева могуће је приступити сигурнијем прототипском развоју или некој врсти симулације. После доношења одлуке о даљем развоју у почетном као и у сваком цилкусу спирале, следи фаза инжењеринга са одабраним моделом развоја софтвера, моделом водопада, моделом прототипског развоја, или било којим другим одабраним. Веома важно за модел спрале, на крају сваког циклуса развоја, корисник оцењује производ и даје сугестије за његову модификацију. На основу повратне спреге од корисника, припрема се фаза планирања новог циклуса развоја и анализа ризика. На основу анализе ризика који је део сваког циклуса може се донети одлука о прекиду даљег развоја или о наставку.

Предности примене спиралног модела су:

- Важни утицајни фактори се откривају раније;

- Софтвер инжењери се укључују у пројекат у раније;

- Анализа ризика - кључна разлика овог модела развоја у односу на остале;

- Велика флексибилност у фази инжењеринга и могуће комбиновање различитих модела;

- Систематски приступ преузет из модела водопада без негативних ефеката;

Недостаци примене спиралног модела су:

- Претерана униформност и конзистентност у развоју;

- Скуп модел за примену у развоју мањих пројеката, посебно јер анализе ризика захтевају укљученост експерата;

- Лоше спроведена анализа ризика може имати мултипликативно дејство у развоју.

2.2.2.6. Модел заснован на компонентама

Основна идеја овог модела је успостављање софтверског производа као организованог скупа појединачних компоненти. Конфигурисање, груписање, специјализовање и документовање компоненти су неке од основних активности у овом приступу. Особине компоненти зависе од њихове величине, сложености и функционалних могућности. Уобичајени приступ организовању компоненти је експлоатација сличности. Ослања се на заједничке структуре података и алгоритмиме за обраду. Сличност међу компонентама може да се заснива и на присуству сличних подсистема. Употреба већ урађених готових или делимично готових компоненти значајно убрзава поступак имплементације.

Развој софтвера уз поновно коришћење компоненти, ако се правилно и рационално примењује, скраћује време израде софтвера, повећава робустност система, подиже квалитет производа захваљујући вишеструкој провери појединачних компоненти, смањује трошкове развоја и значајно олакшава одржавање производа. Може се рећи да је компонентни модел развоја еквивалент објектној парадигми из области програмирања са сличним предностима и манама.

2.2.2.7. Модел унифицираног процеса развоја

Модел унифицираног процеса представљен је 1999. године. Аутори овог модела су: Џејкобсон, Буч и Рамбау (Ivar Jacobson, Grady Booch & James Rumbaugh). Овај модел по својој суштини комбинује итеративни и инкрементални

приступ развоју софтвера. Подразумева интензивно коришћење UML-а, њиховог заједничког обједињеног језика моделовања. Модел је нашао широку примену, омогућио је унапређење продуктивности посебно на великим пројектима. Посебна вредност модела је што највише захваљујући UML-у пружа свим учесницима у развоју, јединствен и заједнички скуп метода и језика за спровођење процеса развоја. Самим тим комуникација међу члановима тима за развој је подигнута на висок ниво, сви имају приступ и разумевање потребним информацијама у облику база знања, пројектним обрасцима, разним упутствима и препорукама и слично. Овај модел није догматски постављен, представља радни оквир који је прилагодљив различитим пројектима у разним окружењима.

Овај модел приказује процес развоја из временске и структурне перспективе. Посматрано из временске перспективе модел описује процес кроз животни циклус развоја софтвера - динамичка димензија модела. Посматрано из структурне перспективе модел описују активности и ефекте активности кроз различите улоге - статичка димензија модела.

Посматрано из временске перспективе предвиђа четири фазе развоја: започињање (inception), разрада (elaboration), конструкција (construction) и транзиција (transition). Ове фазе се свака појединачано извршавају у неколико итерација, при чему тачан број итерација по фазама зависи од конкретног процеса развоја. Итерација свакако укључује активности анализе, дизајна, имплементације и тестирања.

Фаза започињања, или почетна фаза, представља прву и кратку фазу развоја. Она треба да омогући прикупљање основних захтева. Током ове фазе треба да се одреде најважније функционалности система, постави иницијални предлог архитектуре решења. Дефинише се обим и границе система, план реализације, скуп алата и потребни ресурси који ће се користити, идентификују се могући ризици.

Фаза разраде је друга фаза. У оквиру ове фазе дефинише се архитектуре система. Обраћа се значајна пажња на ризике везане за захтеве, архитектуру, трошкове и за конкретне активности у току развоја система. Разрађују се детаљно

кориснички захтеви, спроводи се анализа и дизајн система. У оквиру ове фазе формира се детаљни план пројекта.

Фаза конструкције, као трећа фаза, садржи активности детаљног дизајна, имплементације и тестирања система. Предуслов за успех ове фазе је успешна реализација претходних фаза. Може садржавати и прикупљање евентуално измењених захтева. Спроводи се детаљна разрада архитектуре и активности оперативне израде система, интегрисања компоненти и подсистема као и тестирање система и обуке корисника. Ово је најобимнија фазу и обично заузима бар пола укупног предвиђеног времена за реализацију целог пројекта.

Фаза транзиције значи предају софтвера крајњим корисницима и њихову обуку. У претходној фази корисницима се омогућава пробно коришћење тренутне верзије софтвера, као резултат добијају се повратне информације од корисника о вредности производа.

Посматрано из структурне перспективе модел идентификује шест основних и три помоћне дисциплине. Основне дисциплине су: дисциплина пословног моделовања, дисциплина управљања захтевима, дисциплина анализе и дизајна, дисциплина имплементације, дисциплина тестирања и дисциплина распоређивања. Помоћне дисциплине су: дисциплина конфигурисање и менаџмента променама, дисциплина управљање пројектом и дисциплина окружење.

2.2.2.8. Агилни модели развоја

Агилни развој софтвера је заснован на итеративном и инкременталном моделу развоја. Основни принципи агилне методологије су [10]:

− веома кратки развојни циклуси

− флексибилни самоорганизујући развојни тимови са равном структуром

− пројекте креирају и воде високо мотивисани појединци

− брзо прилагођавање променама захтева

− испорука функционалног софтвера у свакој итерацији

− функционални софтвер као основно мерило напретка пројекта

Проблеми са пробијањем временских рокова и буџета пројеката развоја софтверских производа довели су крајем двадесетог века до појаве нових агилних модела развоја. Све већа сложеност коришћених технологија и повећање корисничких захтева створили су потребу за еластичнијим моделима развоја са већом и активнијом константном улогом корисника у свим фазама и активностима. Основна идеја примене агилних модела је да се минимизира време и трошкови размене информација између свих особа које учествују у развоју. То омогућава брзу повратну спрегу и повратне информације о резултатима донетих одлука и решења у развоју система. Учесницима у развоју и њиховој међусобној комуникацији се поклања посебна пажња, то је изражена особина агилних модела. Агилни процеси истичу јединствене способности појединаца и тимова. Агилни тимови су самоорганизовани и високо прилагодљиви, са интензивном комуникацијом, јединством ка заједничком циљу, узајамним уважавањем и поверењем, заједничким и ефикасним језиком и рутинама у доношењу одлука. У пројекту развоја учествују особе различитих вештина, знања, искуства и способности. У агилним развојним тимовима, компетенције појединаца представљају главни фактор успешности пројекта. Најважније је да су појединци на пројекту довољно квалитетни без обзира на то који модел развоја користи. Ако су чланови развојног тима са недовољним компетенцијама или ако је исказан недостатак корисничке подршке, пројекат развоја који год модел развоја да се користи води ка неуспеху. Агилан тим међутим лоше функционише у оквиру круте организације, стога агилни развој није прикладан за све ситуације. Инсистирање на агилним принципима у некооперативним организацијама није сврсисходно. Познато је да се агилни развој најбоље реализује у мањим тимовима, у специфичним, комплексним и променљивим пројектима. Организациона култура оријентисана на људе и сарадњу представља идеалан амбијент за агилни развој.

Неки од познатих модела агилног развоја су. Extreme Programming (XP), Adaptive Software Development (ASD), Dynamic Systems Development Method (DSDM), Scrum, Feature Driven Development (FDD), Agile Modeling (AM) и други. Посебно популарна агилна методологија је Екстремно програмирање (Extreme programming). Пракса екстремног програмирања даје препоруке [10]:

− стремити једноставном и јасном програмском коду;

− често детаљно анализрати и рефакторисати програмски код;

− програмирати у пару;

− аутоматизовано тестирати комплетан програмски код;

− планирати развој са очекиваним променама корисничких захтева;

− интензивна комуникација са клијентом и између програмера у оквиру тима.

Програмирање у пару значи физичко присуство два програмера у току непосредног писања програмског кода са узајамном провером урађеног. Рефакторисање подразумева измене програмског кода без измене функционалности програма, али са побољшањем дизајна програма Побољшања могу бити у смислу боље читљивости кода или ефикаснијем извршавању - бољим перформансама.

2.2.2.9. Синхронизуј и стабилизуј модел

Синхронизуј и стабилизуј модел (енгл. synch-and-stabilize) је модел развоја који се примењује у ситуацијама када је потребно користити више засебних тимова за развој којима се поверава развој засебних модула или подсистема решења. Ови тимови могу раде паралелно и са високом аутономијом. Модел међутим предвиђа синхронизацију резултата рада појединачних тимова према потребној динамици. Приликом синхронизације потребно је спровести и контролу (стабилизацију) резултата процеса развоја. Обично се формира посебан тим за ове

активности. Како модел дозвољава измене у било којој фази процеса развоја сматра се да је овакав процес развоја флексибилан и прилагодљив променама захтева.

По овом моделу постоје три фазе у процесу развоја:

1. Фаза планирања

Дефинисање визије производа, спецификације и пројекта;

2. Фаза развоја

Развој кроз неколико секвенцијалних потпројеката;

3. Фаза стабилизације

Интерно и екстерно тестирање, коначна стабилизација и испорука производа кориснику;

2.2.2.10. Изгради и фиксирај модел

Изгради и фиксирај модел (енгл. build-and-fix) је наивни модел развоја информационог система. Не препоручује се осим за најједноставније пројекте. Систем се изграђује без одговарајућих спецификација и корака у дизајну. У основи софтвер се креира и модификује изнова све док крајњи корисник не буде задовољан. Трошкови овог приступа могу бити значајно већи него што би били да се спроведу макар основне активности прикупљања захтева и анализе.

2.2.2.11. Комбиновани модели

Горе описани модели су углавном приказивани као алтернативни, а мање као комплементарни модели развоја. У многим ситуацијама модели се могу комбиновати тако да се постигну предности од свих на само једном пројекту. Спирални модел је одличан пример комбиновања два модела. Слично важи и за

модел унифицираног процеса развоја. Сваки модел развоја може послужити као основа са којом се може комбиновати неки други модел. Не треба бити догматичан у избору одређеног модела у развоју софтвера. Природа софтверског производа најчешће диктира који модел развоја треба применити. Комбиновањем модела, резултат постигнут у коначном исходу често ће бити повољнији него што би се постигло применом конкретног модела.

Имајући у виду да је природа софтверског система који се разматра у овом раду систем за обраду података, предложен је комбиновани функцијски оријентисан модел развоја са следећим елементима:

- Развој софтверског производа се ослања на корисничке функције

Корисничке функције представљају основне елементе за успостављање жељеног понашања система, користе се за проверу архитектуре система и за комуникацију између учесника у пројекту;

- Развој се заснива на архитектурном приступу

Архитектура система је основа за концептуализацију, конструкцију, управљање и развијање система током пројектовања;

- Развој се одвија итеративно и инкрементално

Предвиђен је развој и управљање низом функционалних верзија софтвера, при чему свака нова верзија представља инкрементално побољшање претходне верзије софтвера;

Објектни приступ дизајну са друге стране подразумева интеграцију концепта података и концепта функција система. То је приступ где пројектант првенствено размишља о “објектима” а мање о “функцијама” Приступ је постао популаран у 80-тим у 90-тим годинама 20. века. Данас је најчешћи начин имплементације софтвера.

2.3. Моделовање и спецификација информационог система

SSA је методологија која се користи за моделовање и спецификацију информационог система, односно софтвера. Аутори структурне системске анализе описали су њене по много чему напредне особине [11, 12]. SSA у основи поставља у средиште податке и процесе обраде тих података. Систем се посматра из угла података. SSA се користи као методолошки поступак декомпозиције система на подсистеме. Процеси обраде података приказани су дијаграмима токова података.

Методични приступи пројектовању софтвера засновани на структурним методама предложени су у разним облицима и од стране различитих аутора: Структурно Пројектовање - Constatine & Yourdon (1978) [11], Анализа Структурних Система - Gane & Sarson (1979), Џексонов Развој Система - Jackson, (1983), објектно-оријентисано пројектовање - Robinson (1992), Booch (1994), Rumbaugh (1999). Коришћење структурних метода подразумева креирање графичких модела система. Структурни приступ је успешно примењен у великом броју софтверских пројеката. У моделу протока података систем је тако моделован да се посматра као процес трансформације података. Модел ентитет-веза се користи да опише ентитете и везе између њих. Овај модел је уобичајен за описивање структуре базе података. Структурни модели се примењују и за документовање компоненти система и односа између њих.

SSA као средство за моделовање система користи се и у процесу дизајнирања система. Резултат дизајна система треба да буде прецизни опис грађе система. Треба да обезбеди опис од којих се делова систем састоји, какве су везе између делова система, структуру података и на нижем нивоу - детаљнијем нивоу, и алгоритме који ће се користити и интерфејс између система и корисника. Ово је итеративни поступак. Полази се од почетног дизајна на вишем логичком нивоу (дијаграм контекста), па се постепеним увођењем детаља кроз узастопне итерације долази до прецизнијег модела. Спроводи се декомпозиција већих делова система на мање подсистеме и затим њихова даља декомпозиција.

Декомпозиција при дизајну информационих система спроводи се следећи један од следећа два приступа:

- Функционални приступ

Систем се посматра као функција (процес). Састоји се од функционалних целина. Полази се од система као процеса на највишем логичком нивоу. Постепено се процеси декомпонују на све мање процесе чиме се формира хијерархијска структура система. Стање система представља се глобалним репозиторијумом података. Подаци су опште доступни функцијама у систему. Имплементација се реализује уобичајено помоћу класичних програмских језика и процедуралним стилом програмирања. Овакав приступ користи се традиционално у датацентричним системима.

- Објектни приступ

Систем се представља као скуп објеката. Делови система су објекти, а објекти су одређеног типа односно класе. Грађа система је комбинација агрегација и наслеђивања. Имплементација се реализује у неком од многобројних програмских језика који обезбеђују објектно оријентисану парадигму. Стање система остварује кроз стања појединачних објеката.

У оквиру SSA користи се функционални приступ моделовању информационог систма. При развоју информационих сисема са нагласком на једноставности одржавања система добра пракса је пројектовање система засновано на независним и добро дефинисаним компонентама. Лабава међусобна повезаност независних самосталних компоненти решења, добра подела послова међу њима и прецизна дефиниција њихових интерфејса је опис система развијених применом објектног приступа. Ако је фокус приликом креирања информационог система на ефикасности и перформансама система, може се користи дизајн са ограниченим бројем сложенијих компоненти чиме се смањује саобраћај између њих. У таквом систему софтверске компоненте извршавајући своје функције користе централно складиште података и то је врста система која се може добити применом функционалног приступа.

Модели створени у току описивања система даље се разрађују у току фазе дизајна. У почетним активностима модели описује постојећи систем, а поступак дизајна води ка моделирању будућег система. Модели представљају спону између различитих фаза развоја информационог система.

Основни модел који се користи у оквиру SSA је модел протока података. Спада у моделе понашања. Велика већина пословних информационих система су примарно вођени од стране података. Ови системи су условљени уносом података. Разне обраде података чине језгро система и релативно је мало спољних догађаја које систем треба да процесира. Модел протока података је погодан да представи понашање ових система. Системи реалног времена, који садрже велики број интеракција у недефинисаном временског оквиру и редоследу, за које се каже да су вођени догађајима, обично имају процесирања података мањег обима. Модел протока података је интуитиван и олакшава комуникацију са стејкхолдерима и корисницима система. Приказује функционалну перспективу система. Свака трансформација података представља једну функцију или процес.

При моделовању система коришћењем модела протока података обично се користи "од горе ка доле" (енглески top-down) приступ. Могуће је међутим у неким фазама и деловима система применити супротни смер и најпре развити моделе нижег нивоа, а потом их интегисати и абстраховати да креирају модел вишег нивоа апстракције. У top-down приступу, систем се прво дефинише на једном глобалном нивоу. Потом се узастопним итерацијама систем декомпонује на логички ниже једноставније делове. Поступак се понавља све до нивоа примитвних функција које нема потребе даље декомпоновати. Када је неки процес доведен до тог примитвног нивоа, то значи да је та обрада састављена од елементарних корака обраде који се извршавају сукцесивно и могуће их је директно, некада и један-на-један превести у команде програмског језика. На овај начин, врши се декомпозиција система у програмске модуле који изводе тачно дефинисане програмске задатке. Овако дефинисани програмски модули треба да буду логички независини чиме се поједностављује одржавање и даљи развој система.

Дијаграм тока података (ДТП) је средство за креирање формалног описа процеса обраде података. ДТП може да садржи четири основна типа компоненти:

- процес обраде података

- ток података

- складиште података

- интерфејс

Сваки ток података приказан у дијаграму мора да има и извор и одредиште. На једном крају тока података може бити интерфејс, процес или складиште података, а на другом крају мора бити процес. То значи да се једним током података не могу непосредно повезати два интерфејса, два складишта или складиште и интерфејс. Сваки ток података мора имати име. Изузетак је ток података који је повезам са складиштем података. Такав ток података не мора имати име, сматра се да је тада имплицитно именован садржајем складишта података. Ток података може да се грана.

Сваки процес има своје име а које треба да садржи ознаку која говори о позицији у хијерархији процеса. Сваки процес треба да има бар један улазни и бар један излазни ток података. Складиште података може бити без улазног тока или без излазног тока. Сваки интерфејс мора имати бар један улазни или излазни ток података, иначе не би имао смисла.

Дијаграм контекста је дијаграм највишег нивоа и садржи само један процес. Дијаграм контекста као један процес приказује цео систем, садржи и интерфејсе који су ентитети ван посматраног система, као и токове са њима.

Процеси који се даље не декомпонују су примитивни процеси. Ови процеси се не декомпонују, али је потребно приказати логику њиховог рада кроз мини-спецификацију система. То може бити блок дијаграм или псеудокод којим се приказује секвенца извршавања операција.

Осим процеса, потребно је декомпоновати токове података и и складишта података. Улазни и излазни токови на дијаграмима нижег нивоа добијених

декомпозицијом неког процеса морају бити у складу са са улазним и излазним токовима тог процеса приказаним на дијаграму вишег нивоа. Складишта података која се приказују на одређеном нивоу морају се после појављивати на свим нижим нивоима. Дијаграм тока података не треба да буде превише сложен, правило је да добар дијаграм треба да садржи максимално 7-8 процеса.

2.4. Технике надгледања базе података

Базе података су данас обавезни део система који покривају све могуће аспекте нашег живота. Наши банковни рачуни, медицински картони, евиденције о запослењу, телефонски разговори - готово свака информација о нашим животима налази се сачувана у неком модерном систему за управљање базама података. Ако је база података или систем у којем је сачувана база података, на неки начин несигурна, потенцијалне последице на наше животе па чак и последице на шире окружење, могле би бити погубне [13]. Како се информациони системи засновани на базама података успостављају као кључни за управљање, одлучивање и дневно функционисање разних организација, тако сигурност база података и исправност података у њима постају пресудни.

Надгледање базе података је скуп активности које омогућавају надгледање, праћење и бележење свих важних догађаја у систему базе података. То може бити промена поставки сервера, пријављивања корисника на систем или рад са подацима, промена структуре података или манипулација самим подацима.

Разлози за надгледање базе података могу бити различити: спречавање нежељених активности, побољшање ефикасности система, откривање корисних информација о раду система, и слично. Некада постоји и законска регулатива која налаже спровођење надзора учесника у систему. Такав надзор може омогућити регистровање одређених активности и поништавање њиховог ефекта. Свако нарушавање сигурности података може умањити поверење корисника у организацију.

Уобичајени начини надгледања базе података су надгледање помоћу интерних механизама базе података, праћењем мрежног саобраћаја и кроз апликативни слој система. Сваки од приступа има своје предности и недостатке, а могуће је користити и комбинацију ових модела. Могуће је и скенирање и анализирање дневника трансакција базе података. Сваки систем за управљање базом података (енглески Database management system - DBMS) може да користи механизам дневника трансакција за бележење сваке измене у бази података. Основна сврха дневника трансакција је могућност опоравка података у случају краха система. Као посебна компонента DBMS-а обично постоји софтвер који

тумачи записе у дневнику трансакција. Осим чињенице да је могуће искључити ову опцију, односно подесити систем тако да не снима дневник трансакција, треба знати да се по правилу selekt команда не бележи у дневнику трансакција.

Надгледање помоћу интерних механизама базе података је најчешћи начин надгледања. Реализује се коришћењем разних елемената самог система за управљање базом података, као што су различити системски дневници (енглески Log) базе података и механизам окидача. Дневници су најбољи извор детаљних информација о активностима и догађајима сервера базе података. Постоји више метода за надгледање активности и догађаја које се одвијају на серверу базе података. Могуће је пратити активности од значаја на нивоу целог система базе података и на нивоу појединачних база података. Могуће је и потребно је сузити обим активности и објеката базе који се прате пошто би неселективно праћење резултирало великом, огромном количином информација које би биле тешко а могуће и практично немогуће за руковање и анализу. Подаци потребни за надгледање могу се снимати интерно у самом систему, али и у екстерну датотеку у фајл систему тог или дистрибуираног сервера. Такође, могуће је такве метаподатке третирати као обичне податке и сместити их директно у стандардну Sql табелу.

Окидачи (енглески Trigger) се често користе као техника надгледања SQL база података. ИСО стандард - ISO/IEC 9075 дефинише окидач као спецификацију активности која се мора извршити сваки пут када се одређена операција изврши на одређеном објекту базе података. Окидач представља блок Sql кода или блок команди процедуралног језика који се користи у оквиру система базе података, који се аутоматски извршава када корисник или систем покуша или изврши модификацију података у оквиру одређене табеле у бази података. Најчешћа примена је за очување интегритета података у бази података. У зависности од тога које операције изазивају покретање окидача могу се поделити на: INSERT, UPDATE и DELETE окидаче. Покретање или окидање окидача може се догодити непосредно пре (BEFORE) или после (AFTER) извршавања активности за коју је везан окидач. Окидачи могу да се активирају када се дешавају промене на нивоу

табеле али могу да се активирају и приликом ажурирања сваког појединачног реда табеле. У неким системима окидачи могу да буду повезани не само са командама за модификацију подацима - DML командама (енглески Data Manipulation Language) већ и са командама за дефинисање и измену објеката у бази - DDL командама (енглески Data Definition Language). Ове нестандардне окидаче имају само неки системи база података и омогућавају креирање напредних метода за надгледање базе података. Многи аутори софтверских решења који користе SQL базе података, изразили су потребу за SELECT окидачем [14]. То би био окидач који би се извршио када се догоди SELECT команда над објектом базе који се посматра. Таква врста окидача није до сада била омогућена у већини система за базе података мада се врше истраживања на ову тему [15]. Сматра се да би постојање selekt окидача највише користило за форензичку анализу база података. Постојање окидача који нису везани за податке, као што је на пример окидач који се активира приликом пријављивања на систем (енглески - login trigger) може бити безбедносна претња - основа за потенцијални DDOS (Distributed Denial of service) напад.

Окидачи се користе за:

- надгледање промена у бази података
- спречавање и поништавање промена
- наметање дефинисаних ограничења
- меморисање историјских вредности података
- аутоматско ажурирање дневника
- примењивање пословне логике

Употреба окидача има неколико предности. Релативно је једноставно имплементирати их, друго, флексибилни су и могуће их је мењати, модификовати и деактивирати или уклонити у било ком тренутку. Окидачи зависе од структуре података и у случају измене структуре података морају и сами бити прилагођени па су стога захтевни за одржавање. Осим тога употреба окидача је компликована када се користе комбинације повезаних окидача, у трансакционим обрадама и код дистрибуираних база података. Окидачи нису погодни за спречавање унутрашњих претњи од корисника са одговарајућим правима у бази. Осим тога окидачи увек

имају негативан утицај на перформансе система. Окидачи не прихватају параметре и аргументе и могу мултиплицирати грешке ако су неправилно написани.

Надгледање преко мрежног саобраћаја за разлику од дневника трансакција који је интерни механизам базе података може бити потпуно независан поступак од саме базе података. Надгледањем и анализом мрежног саобраћаја може се надгледати база података потпуно независно од базе података и без могућности детектовања активности од стране система за управљање базом података. Овакав приступ нема утицаја на перформансе базе и не подлеже ограничењима система за управљање базом података. Ипак, постоје активности у свакој бази података које нису иницирane спољним захтевима и могу бити релевантне за посматрани систем, а оваквим приступом остају ван видокруга посматрача. Такође, овакав приступ није могућ у случају комуникације са базом података преко заштићеног криптованог канала. Аутори у [16] предлажу механизам за евидентирање и надгледање активности базе података помоћу праћења мрежног промета. Архитектура предложене шеме садржи три главне компоненте: снимање пакета који иду ка и од базе података, анализирајући комуникацијске протоколе рашчлањивање пакета и на крају употреба преведених пакета за коначно надгледање структура података у бази.

Различити системи база података имају различите интерне механизме надгледања али већина укључује надгледање следећих догађаја:

- Успешне пријаве и одјаве са система
- Неуспешне пријаве и одјаве са система
- Рестарт сервера базе података
- Команде администратора система
- Insert, update и delete операције
- Извршавање ускладиштених процедура (енглески Stored procedure)
- Покушај кршења референцијалног интегритета података
- Покушај кршења интегритета података непоштовањем ограничења
- Промене у системским табелама и речнику података

Осим надгледања базе података интерним механизмима и надгледањем мрежног саобраћаја, могуће је и проактивно надгледање базе података из апликативног слоја решења. Таква техника заобилази проблеме као што су: надгледање активности привилегованих корисника, оштећење базе података или дневника трансакција базе података, дистуиране базе података и слично, који могу представљати проблем уобичајеним методама надгледања. Овакав начин надгледања базе обично подразумева модификовање и додавање објеката у посматрану базу података. Апликативни приступ за надгледања базе погодан је за праћење перформанси система. Робусно решење за спољни систем надгледања требало би да обезбеди засебан механизам за складиштење података о надгледаном систему и требало би да буде одвојен политиком безбедности од привилегованих корисника посматране базе.

2.5. Рачунарски подржан развој софтвера

Рачунарски подржан развој софтвера (енглески: Computer Aided Software Engineering - CASE) представља развој софтверских производа уз помоћ рачунара. CASE означава сваки софтверски производ који помаже при реализацији било које активности у процесу развоја софтвера. CASE технологије су намењене аутоматизацији процеса у току развоја софтвера. CASE је акроним који је први пут коришћен од стране Nastec Corporation 1982. године.

CASE технологије су настале као одговор на потребу да се унапреди, аутоматизује и повећа продуктивност при развоју софтверских производа. Основна идеја је примена софтвера за подршку при развоју софтвера. CASE технологије покривају широку област од појединачних алата за конкретне послове до великих софтверских система за подршку комплетним методологијама развоја софтверског производа. CASE технологије не представљају методе развоја, оне су само средство, али често у себи садрже базе знања о методама и техникама које имплементирају. Самим тим организације које су успешно примењивале CASE технологије су такође примењивале и методологије које су органски повезане са софтверским алатима. Ове технологије су сада системи који интегришу хардвер, софтвер, базе података, људе и процедуре. Процедуре се називају CASE методологијом, а база података CASE енциклопедијом.

Средином деведесетих година двадесетог века, повећањем употребе објектно-оријентисаног моделовања у односу на функционално моделовање приликом развоја софтверских производа, традиционални CASE алати уступају место новим CASE алатима који подржавају UML. Већина CASE алата који су сада присутни на тржишту обезбеђују подршку за развој објектно оријентисаних софтвера и користе UML. У табели 1, дат је приказ неких актуелних UML CASE софтвера према [17].

Неки тренутно доступни CASE алати подржавају Agile Development као и Model Driven Engineering (МДЕ) и xUML. Примећено је да сви алати који подржавају MDE пружају аутоматизацију за трансформацију модела што је главна активност еволуције вођене моделима [18].

Табела 1: Lista UML CASE софтверских alata [17]

Назив	**Произвођач**	**Платформа**	**Најнов. издање**	**Open source**	**Програмски језик**
Astah	Change Vision, Inc.	Cross-platf. (Java)	2019-01-30	No	Java
BOUML	Bruno Pagès	Cross-platform	2020-03-01	No	C++/Qt and Java ("plug-out")
Cacoo	Nulab	Windows 7+, Mac OS X	July 2018	No	HTML5
CaseComplete	Serlio Software	Windows	2013-04	No	C#
Dia	Alexander Larsson	Cross-platf. (GTK+)	2012-07-05	Yes	C
Eclipse UML2 Tools	Eclipse Foundation	Cross-platf. (Java)	2018-12-03	Yes	Java
Edraw Max	Edrawsoft	Windows, Linux, macOS	2015-03	No	C++
Enterprise Architect	Sparx Systems	Windows, Linux, macOS	2019-03-06	No	C++
Gliffy	Gliffy	Internet browser	2015-01 (v. 5.1)	No	HTML5 & JavaScript
JDeveloper	Oracle Corporation	Cross-platf. (Java)	2019-09-27	No	Java
Microsoft Visio	Microsoft	Windows	2016 (v16.0)	No	Unknown
Microsoft Visual Studio	Microsoft	Windows	2016-06-27	No	C++, C#
Modelio	Modeliosoft	Windows, Linux, macOS	2020-02-03	Yes	Java
MyEclipse	Genuitec	Windows, Linux	Unknown	No	Java
NetBeans	Oracle Corporation	Windows, macOS, Linux, Unix	2013-2020-06-04	Yes	Java
Open ModelSphere	Grandite	Cross-platf. (Java)	2009-11-04	Yes	Java

Papyrus	Commissariat à l'Énergie Atomique	Windows, Linux, macOS (Java)	2018-12	Yes	Java
PlantUML	Arnaud Roques	Cross-platf. (Java)	2019-09-22	Yes	Java
PowerDesigner	Sybase	Windows	2018	No	Unknown
PragmaDev Studio	PragmaDev	Windows, Linux, OS X	2018-02-07	No	Python, C, C++
Prosa UML Modeller	Insoft Oy	Windows	2013-10-19	No	C/C++
Rational Rhapsody	IBM	Windows, Linux	2019-12-15	No	C, C++, Java, Ada
Rational Software Architect	IBM	Windows, Linux	2015-09-18	No	Java/C++
Rational System Architect	IBM	Windows	2013-03-15	No	Unknown
Reactive Blocks	Bitreactive	Windows, macOS, Linux	2016-09-16	No	Java
Software Ideas Modeler	Dusan Rodina	Windows (.NET), Linux	2020-01-20	No	C#
StarUML	MKLab	Windows, macOS, Linux	2018-08-17	No	Delphi
Umbrello UML Modeller	Umbrello Team	Unix-like; Windows	2019-12-18	Yes	C++, KDE
UML Designer	Obeo	Windows, macOS, Linux	2019-01-29	Yes	Java, Sirius
UMLet	The UMLet Team	Windows, macOS, Linux	2018-08-05	Yes	Java
UModel	Altova	Windows	2019-10-9	No	Java, C#, Visual Basic
Umple	University of Ottawa	Cross-platf. Java/Eclipse	2018-02-19	Yes	Umple, Java, PHP, Javascript

Основни циљеви примене CASE технологије су: скраћење времена израде пројеката, повећање продуктивности пројектаната, повећање квалитета софтвера, унапређење перформанси система, контролисана примена процедуре развоја и слично. Да би се успешно применио неки CASE систем, претпоставка је усвајање одговарајуће методологије развоја софтверског производа. CASE алати су средства у рукама софтвер инжењера.

Постоји више могућих начина класификације CASE технологија и алата у зависности од критеријума који се посматра. То може бити класификација CASE технологија/алата на основу функционалности које се обезбеђују употребом истих, фазе софтверског процеса у којима се примењују или врсте хардвера и софтвера за које су предвиђене.

У зависности које фазе процеса развоја софтвера покрива CASE технологије се деле према [19] на:

- Пројектански CASE, аутоматизују се почетно фазе развоја: планирање, анализу и дизајн,

- Програмерски CASE, аутоматизују се наредне фазе развоја: програмирање, имплементацију и експлоатацију и одржавање,

- Интегрисани CASE, подржава све фазе развоја система.

Према функцији коју аутоматизују, CASE алати се могу поделити на:

- алате за анализу и пројектовање

- алате за пројектовање базе података

- алате за програмирање

- алате за одржавање софтвера

- алате за управљање пројектима

У [20] Fuggetta предлаже класификацију у три категорије:

1. Алати подржавају индивидуалне задатке као што су провера конзистенције пројектовања, компајлирање програма, упоређивање тест резултата, итд. Алати могу бити генералне намене, самостални алати или могу бити груписани у радне тезге,

2. Радне тезге (workbenches) подржавају фазе процеса или активности као што су спецификација, пројектовање, итд. Оне се састоје од интегрисаног скупа алата,

3. Окружење подржава највећи део софтверског процеса. Укључује неколико различитих интегрисаних радних тезги.

При развоју CASE технологија доста је пажње посвећено аутоматизацији почетних фаза развоја софтвера зато што се показало да грешке направљене у почетним фазама највише коштају. Примена CASE технологија доприноси квалитету софтверског инжењеринга, штеди време, штеди ресурсе, и омогућава реализацију пројеката који би иначе били тешко изводљиви. Продуктивност се повећава, а грешке у развоју софтвера се значајно смањују. Развој јединствених CASE алата је некада важан део планирања великих и сложених софтверских пројеката. Примена ових технологија доприноси стандардизацији произведеног софтвера.

CASE технологија је сада доступна за већину рутинских активности у софтверском процесу. Резултати примене довели су до побољшања процеса развоја софтвера. Треба разумети да је софтверски инжењеринг као активност пројектовања један креативан процес. CASE системи аутоматизују рутинске активности. Када се буду успешно примениле технологије вештачке интелигенције са подршком процесу пројектовања софтвера то ће довести до већег степена побољшања.

Неки од захтева који се постављају пред будуће CASE технологије су: стандардизација софтверских производа, примена инструкција на природном језику човека, широка примена општих пројектних шаблона, ефикасно управљање пројектом развоја софтвера, успешно и правовремено одржавање система, ефикасно управљања ресурсима, и слично. Да би се обезбедило испуњавање

оваквих захтева, потребна је интеграција са техникама и методама вештачке интелигенције. Коришћење интелигентног CASE софтвера у процесу развоја софтвера и употреба вештачке интелигенције у модулу за кодирање у псеудо код, део су предложеног модела у истраживању само-модификујућег софтверског система [21].

Интелигентне CASE технологије су будућност. Оне треба да омогуће развој софтвера за уз минимално учешће човека.

2.6. Основна разматрања о FMEA

Анализа ризика је поступак који треба да се реализује током развоја производа и током његове експлоатације. Једна од најшире коришћених метода за анализу ризика је Failure Mode and Effects Analysis (FMEA) [22]. Први пут је примењена у компанији Ford Motor и од тада је постала есенцијалана методологија за анализу ризика у аутомобилској индустрији. У литератури може да се нађе велики број истраживања у којима је приказана примена ове методе у ауто индустрији [23, 24]. У овом истраживању са идејом повезивања фазе одржавања софтвера и анализе грешака и ризика, посебно су била од значаја ранија истраживања која повезују одржавање система и FMEA. Поповић и остали, предлажу модификацију FMEA за одабир концепта одржавања, дозвољавајући да опрема буде класификована према значају за конкретну организацију и да заједно са осталим економским и техничким факторима омогући одлучивање о нивоу одржавања и распореду ресурса [25].

FMEA је постала део стандарда ISO/TS 16949 [26]. У конвенционалној FMEA, свака идентификована грешка је оцењена у смсислу три фактора ризика (РФ-а): озбиљност последице која настаје услед реализације грешке (С), вероватноће настајања грешке (О) и могућности откривања грешке (Д). Вредности ових РФ-а је процењена од стране FMEA тима који користе скалу мера [1-10]. Вредност 1 означава да је озбиљност последице и вероватноћа настајања грешака готово занемарљива односно могућност откривања грешке мала. У супротном вредност 10 означава да су озбиљност последице и вероватноћа настанка последица као и могућност откривања грешке велика. Коришћена скала мера је дефинисана на основу података из евиденције и треба напоменути да препоруке конвенционалне FMEA могу се применити само у ауто индустрији. Ранг грешака се добија према индексу који је означен као број приоритета ризика (РПН). Вредност овог индекса се рачуна као производ вредности РФ-а на нивоу сваке идентификоване грешке. На првом месту у рангу налази се она грешка којој је придружена највећа вредност РПН и ова грешка има највећи приоритет у процесу побољшања производа или процеса.

Примена FMEA у другим индустријама захтева од FMEA тима да развије нове скале мера које одговарају разматраном проблем. У литературе може да се нађе релативно мали број радова у којима се разматра проблем анализе грешака у домену информационих технологија применом FMEA методе [27, 28, 29].

Одређивање приоритета ризика користећи конвенционалну FMEA методу има бројне недостатке који су анализирали многи аутори у литератури. Неки од најважнијих недостатака ове методе су: (1) дефинисање мерне скале је засновано на интуитивним претпоставкама, другим речима поставља се питање да ли је дефинисана мерна скала на интервал [1-10] одговарајућа, (2) познато је да доносиоци одлука веома тешко исказују своје процене користећи нумеричке скале, доносиоци одлуке боље исказују своја мишљења користећи лингвистичке исказе, (3) релативна важност РФ-а нема једнаку важност [30, 31, 32], (4) не постоји јасно математичко објашњење зашто је критеријум класификације дефинисан као производ три разматрана РФ-а [33. 34, 35], (5) грешке које имају исту вредност РПН имају исту јачину утицаја иако се разликују вредности РФ-а; овако решење се не поклапа са резултатима који су добијени у реалним системима [33, 34, 35].

Неки аутори су уложили напоре да многе недостатке FMEA методе елиминишу. Кратка ретроспектива ових радова надаље је приказана.

Вредности РФ-а су описани лингвистичким исказима РФс који могу да буду моделовани применом различитих области математике. Као на пример применом теорије вероватноће у [36]; ови аутори су лингвистичке исказе моделирали случајном променљивом са Гаусовом расподелом.

Развој теорије фази скупова [37, 38] је омогућио да се многе неизвесности и непрецизности квантитативно опишу на довољно добар начин. Када се разматра проблем моделовања неизвесности прво питање се односи на избор броја лингвистичких термина којима се описују разматране неизвесности. Углавном, број неизвесности одређују доносиоци одлука на основу свог знања и искуства

респектујући величину проблема. Неки аутори сматрају да човек може највише истовремено да разматра највише седам различитих ставки тако да се сматра да свака неизвесност може да се опише помоћу највише седам лингвистичких термина. Друго питање је избор функције расподеле могућности фази бројева којима се квантитативно описују лингвистички искази. У литератури, могу да се нађу многи радови у којима се различите врсте неизвесности и непрецизности моделирају тип-1 фази скуповима: (а) троугаоним фази бројевима [24, 39, 40], (б) трапезоидним фази бројевима, (ц) интервалним интуитивним фази скуповима [41], (д) хеситант фази скуповима [42]. Најчешће се користе троугаони и трапезоидни фази бројеви којима се на довољно добар начин описују неизвесности а истовремено обим рачунања је значајно мањи него ако се користе фази бројеви вишег реда.

При изучавању неизвесности многи аутори су сматрали да тип-1 фази бројеви не могу довољно добро да опишу неизвесност. Стога, Zadeh (1975) [43] је увео тип-2 фази скуп који представља проширење тип-1 фази броја. Тип-2 фази бројеви су описани помоћу две функције расподеле могућности тако да је омогућен већи степен слободе и флексибилности при описивању неизвесноти. Респектујући ову особину тип-2 фази броја, може да се каже да ова врста фази броја боље описује неизвесносзт него тип-1 фази број. Тип-2 фази бројеви захтевају сложенији обим рачунања тако да је њихова применљивост мања него тип-1 фази бројева. Mendel [44] је увео концепт интервалног тип-2 фази броја. Код ових фази бројева горња и доња фунција расподеле могућности има исту вредност за модалне вредности у домену. Различите неизвесности су моделиране помоћу тип-2 интервалних фази скупова [45].

Треће питање је дефинисање домена фази бројева. Треба нагласити да не постоје правила нити препоруке како изабрати домен на којима су дефинисани фази бројеви. Најчешће се користи стандардна скала мера [46] која је дефинисана на скупу реалних бројева на интервалу [1-9]. Многи аутори су тип-1 фази бројеве дефинисали на интервалу [1-5], интервалу [0-1] [41, 42], на интервалу [0-10], і [1-10] [36].

Као што је раније наведено, релативна важност РФ-а према којима се оцењују грешке имају различиту релативну важност. Тежина РФ-а у општем случају може да се одреди на два начина. Први, јесте директна процена [39, 47]. Ови аутори сугериши да одређивање тежине РФ-а треба да буде постављен као фази групни проблем одлучивања. Сваки доносилац одлуке користи један од унапред дефинисаних лингвистичких исказа којима исказује свој став о тежини РФ-а. Ови лингвистички искази су моделирани троугаоним фази бројевима који су дефинисани на интервалу [0-1]. Агрегација индивидуалних процена у групни консензус је извршена према процедури која је развијена у [39] и коришћењем фази средње Фази отежаног оператора у [47]. Други начин је постављање матрице парова упоређења релативне важности РФ-а. Фази матрица шарова упоређења релативне важности РФ-а је постављена у [28, 45, 48]. Елементи ове матрице су описани троугаоним фази бројевима у [48], тип-2 трапезоидним фази бројевима [45] и прецизним бројевима који припадају интервалу [1-9] у [28]. Вектор тежине РФ-а у [48] је израчунат применом методе проширене анализе [49]. У [30] су вектор тежина одредили применом Shannon's концепта ентропије предложене у [50]. У [28] су користили Best-Worst методу за одређивање вектора тежине РФ-а.

У овом докторском раду релативна важност РФ-а је постављена помоћу фази матрице парова упоређења са интервалним тип-2 трапезоидним фази бројевима као што је предложено [51, 52]. У [51] су тежину разматраних ентитета одређивали применом поступка за поређење тип-2 фази бројева. У [52] су предложили нов поступак за дефазификацију који на бољи начин тип-2 фази број описује скаларном вредношћу. У овом раду усвојен је поступак дефазификације и одређивања вектора тежина као што је предложено у [52].

Одређивање приоритета респектујући многе аттрибуте као и њихове тежине може да буде постављено као задатак више-критеријумске оптимизације (ВКО). У литератури постоји велики број развијених ВКО метода које су развијене на различитим логичким и математичким основама. Стога, може да се каже да резултате добијене применом различитих ВКО метода нема смисла

поредити. Надаље је дата анализа радова у којима су коришћене различите методе за анализу грешака. Једна од ВКО метода која је лако разумљива и применљива је COmplex PRoportional Assessment (COPRAS) која је представљена од стране [53]. Респектујући мишљење многих аутора [54, 55, 56], COPRAS метода је лака за разумевање и веома применљива за налажење оптималног решења добро структурираних проблема и користи се за рангирање грешака софтвера у овој докторској тези.

У [39] су разматрали процену ризика унутар неизвесног окружења. Отежана фази матрица одлучивања је трансформисана у матрицу одлучивања. Елементи фази отежане матрице одлучивања су трансформисане у crisp вредности у поступку дефазификације. Ови аутори су користили методу момента као методу дефазификације [37]. Ранг грешака је добијен применом методе VIKOR која је развијена у [57].

У [48] су користили фази TOPSIS за рангирање грешака. Фази TOPSIS која је коришћена у анализираном раду је развијена у [58]. У [28] су користили конвенционалну TOPSIS за одређивање грешака које су идентификоване у једном софтверу. Позитивно идаелно решење (PIS) и негативно идеално решење (NIS) је одређено према [59]. Дистанце од PIS и NIS су одређене као Еуклидове дистанце.

У [30] су разматрали проблем анализе и рангирања грешака процеса производње. Ови аутори поред уобичајних РФ-а који се разматрају у конвенционалној FMEA уводе и неке друге РФ-а као на пример: фактори окружења, економске факторе, временске факторе, факторе радне снаге и др. У овом докторском раду, ранг идентификованих грешака развијеног софтвера је добијен применом конвенционалне COPRAS методе [53]. Може се сматрати да је ова метода математички једноставнија него остале ВКО методе и да су добијени резултати довољно добри за даљу анализу.

Компаративна анализа одређивања релативне важности РФ-а и ранга РФ-а је приказана у Табели 2.

Табела 2: Компаративна анализа за одређивање

тежине РФ-а и ранга грешака

	Тип променљиве	Број унапред дефинисаних лингвистичких исказа	Домени неизвесних бројеви	Одређивање тежине РФ-а	Рангирање грешака применом ВКО
Liu et al, 2012 [39]	Троугаони фази бројеви	7	[0-1]	Фази групно одлучивање; Нова процедура за агрегацију	VIKOR
Kutlu and Ekmekcioglu, 2012 [48]	Троугаони фази бројеви	9	[1/3-3].	ФАХП развијена у Chang, (1996) [49]	Фази TOPSIS који је предложен у (Chen, 2000 [58]
Adhikary et al. 2014 [30]	Прецизни бројеви	-	-	Shannon концепт ентропије	COPRAS
Peko et al, 2018 [28]	Прецизни бројеви	-	-	Best-Worst метода (Rezaei, 2015) [60]	TOPSIS
Предложени модел	Тип-2 трапезоидни фази бројеви	3	[1-5]	АХП са тип-2 трапезоидним фази бројевима	COPRAS

У табели 2 приказана је компаративна анализа радова који могу да се нађу у литератури у којима је разматран проблем оцене грешака према FMEA оквиру и рангирање грешака помоцу MCDM-ВКО. Многи аутори сматрају да озбиљност последица које настају услед материјализације грешака, фреквенција појављивања грешака, и начин откривања грешака (надаље улазних променљивих) могу довољно добро да се опису помоцу лингвистичких исказа који су моделирани помоћу троугаоних фази бројева (TFNs) [39] и [48].

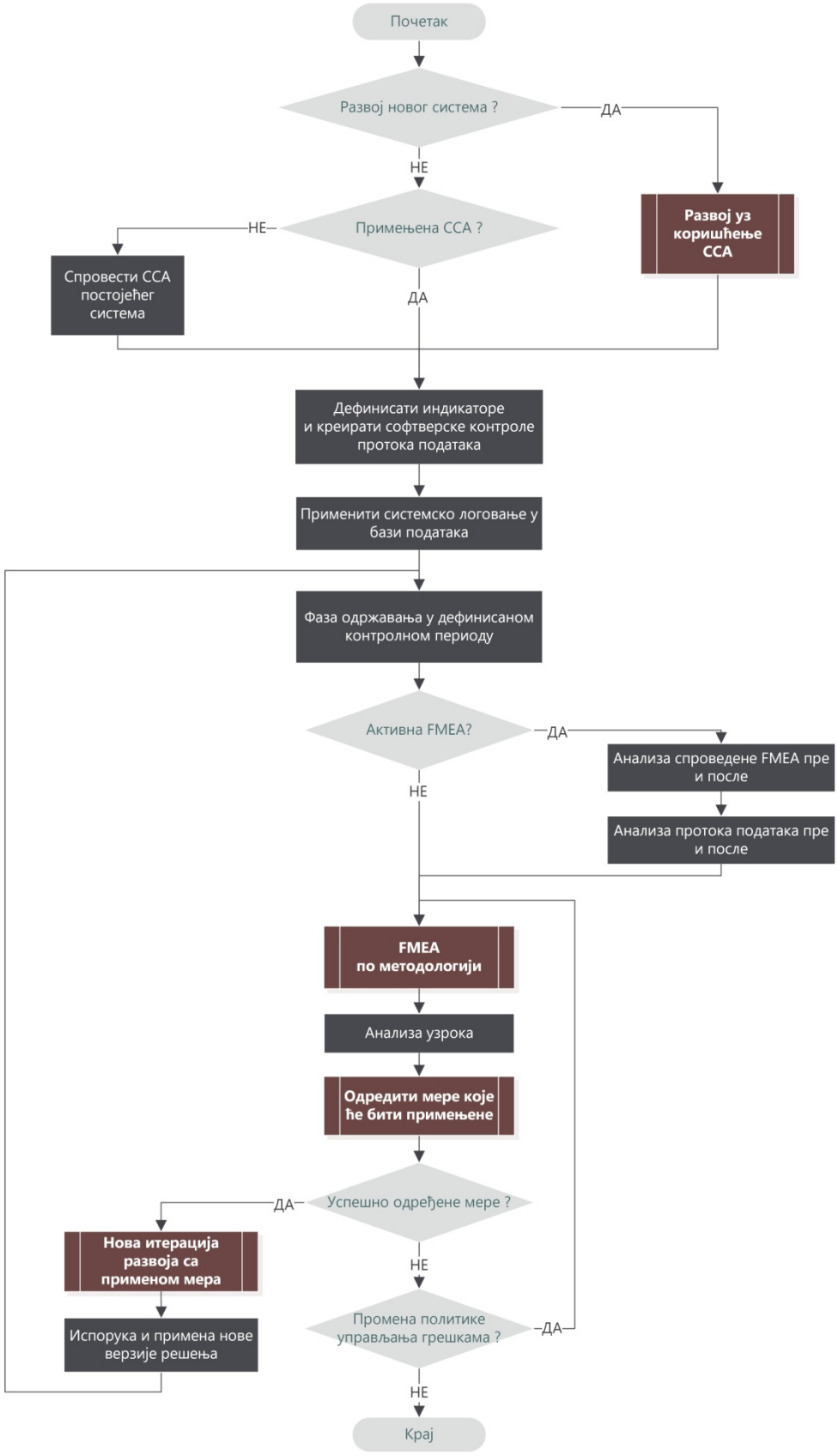
Неки аутори сугеришу коришћење прецизних бројева као што је предложено у конвенционалној FMEA [28, 30]. У овој докторској тези уведена је претпоставка да вредности неизвезних улазних променљивих могу довољно добро да се опишу помоћу интервалних тип 2 трапезоидних фази бројева (IT2TrFNs). Аутори користе различити број лингвистичких исказа помоћу којих описују вредности неизвесних улазних података. Треба напоменути да број лингвистичких исказа зависи од величине и сложености проблема. У свим анализираним радовима претпостављено је да релативна важност улазних података није једнака и да може да се одреди на различите начине. У [39] је проблем одређивања тежине критеријума постављен као проблем групног одлучивања. Агрегиране вредности тежина добијене су применом нове процедуре за агрегацију која је развијена у овом раду. У [48] је постављена фази матрица релативног односа улазних променљивих које су моделиране помоћу TFNs. Домени ових фази бројева припадају интервалу [1/3-3].

Вектор тежине је добијен применом методе сопствене вредности [49]. На овај начин тежине улазних променљивих су описане ординалним прецизним бројевима. На сличан начин је задата релативна важност улазних променљивих у проблему који се разматра у овој докторској дисертацији. Елементи матрице парова упоређења описани су помоћу IT2TrFNs чији домени припадају интервалу [1-5]. Вектор тежине добијен применом процедуре која је развијена у [52]. У осталим разматраним радовима тежина улазних података израчуната је на различите начине. Тако на пример у [28] и у [30] se сматра да тежина улазних променљивих може довољно тачно да се израчуна применом Best-worst методом [60]. Ранг грешака добијен је применом различитих ВКО метода. Може да се каже да не постоји правило на који начин изабрати методу што се респектује да различити аутори предлажу различите методе. Тако на пример у [39] је коришћена Vikor метода. Фази Topsis односно Топсис је коришћена од стране аутора у [48] и у [28], респективно. COPRAS метода је коришћена од стране [30] као и у овој докторској тези.

3. МЕТОДОЛОГИЈА ЗА ОПТИМИЗАЦИЈУ ЕФЕКАТА ПРОТОКА ПОДАТАКА У ИС

Сваки информациони систем (ИС) је јединствен скуп компоненти за прикупљање, чување, обраду и преношење информација. Компоненте ИС-а су хардвер, софтвер, базе података, комуникациони системи. Пословни ИС-и типично обрађују велику количину података који су потребни за одређену организацију. Ови системи представљају технолошке платформе и омогућавају интегрисање и координацију пословних процеса. Софтвер за подршку пословању је у центру оваквих информационих система. Такав софтвер треба да буде поуздан, проширив, повезив са другим системима и да има потребне перформансе. Пословни софтвер може да се развија у оквиру и са ресурсима организације која ће га и користити. Чешћи је случај да организације набављају софтверска решења од специјализованих софтверских компанија које се баве израдом софтвера. Предложена методологија односи се на пословне информационе системе где је могуће тесно повезати процес израде и процес одржавања софтвера.

Елементи предложене методологије од посебног значаја су: Итеративни софтверски процес, Структурна системска анализа, Ефекти протока података, Имплементација надгледања система, Нов хибридни приступ у анализи ризика у фази одржавања софтвера, CASE алати потребни и готово неопходни. На слици 2, приказан је алгоритам методологије у облику дијаграма тока.

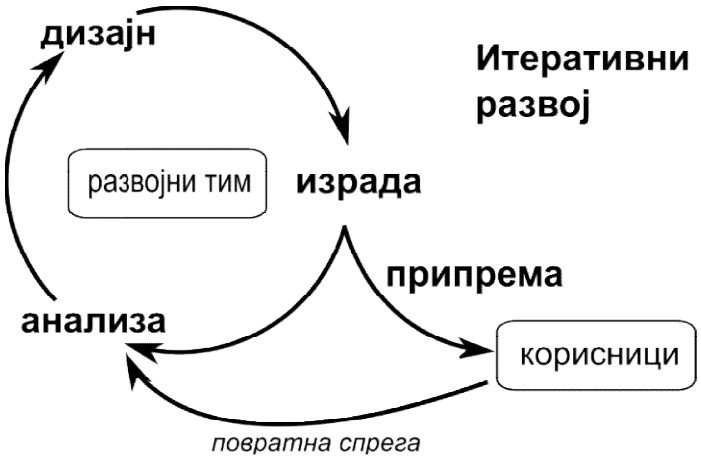


Слика 2: Методологија

3.1. Итеративни софтверски процес

Процес развоја софтвера садржи фазе и различите активности током развоја софтверског производа. Постоји много модела процеса развоја софтвера. Сви они на неки начин организују и повезују активности током софтверског процеса. Основни модел процеса, секвенцијални или модел водопада садржи следеће фазе: Спецификација захтева, Анализа, Дизајн, Имплементација, Тестирање, Испорука и одржавање [4].

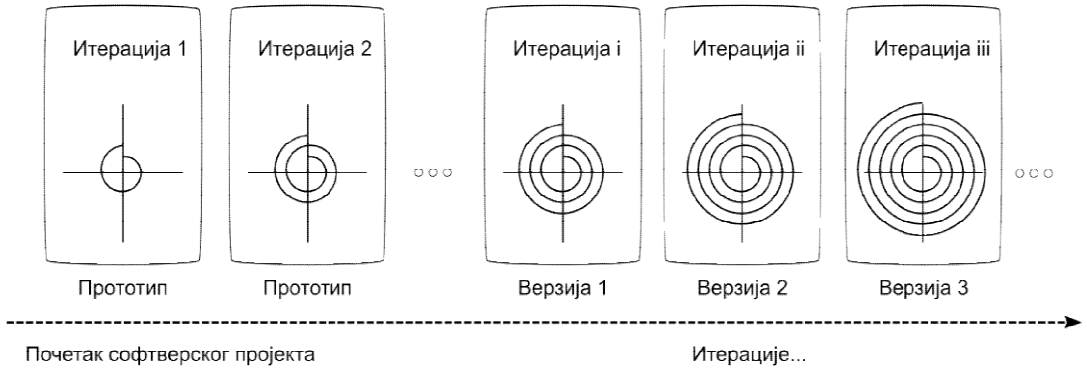
Значајан проблем код развоја софтвера је немогућност сагледавања свих проблема који ће се јавити при коришћењу у раним фазама развоја. Пракса показује да се део захтева може сагледати тек када систем почне да се користи. Проблем који се јавља код класичног развоја софтвера је то да једноставно није могуће предвидети све проблеме у раду [8].



Слика 3: Итеративни модел развоја

Предложена методологија подразумева, независно од конкретног примењеног модела развоја софтвера, итеративни начин развоја. Могуће је у свакој итерацији - циклусу развоја, спровести активности из свих или појединих фаза оригиналног модела, као што је приказано на слици 3. Итеративни развој обично значи и инкрементални развој, што значи да се у свакој итерацији могу додавати нове компоненте које обезбеђују или побољшавају функционалности решења, као што је приказано на слици 4. По завршеној изради и примени сваке нове верзије решења, системски се прикупљају подаци о његовом функционисању и грешкама у експлоатацији. Прикупљене информације у фази одржавања

софтерског решења користе се за планирање даљег развоја. Основна идеја је да се на бољи начин повежу претходне фазе са фазом одржавања, како би се кроз модификовани софтверски процес остварили жељени ефекти протока података у оквиру процеса система.



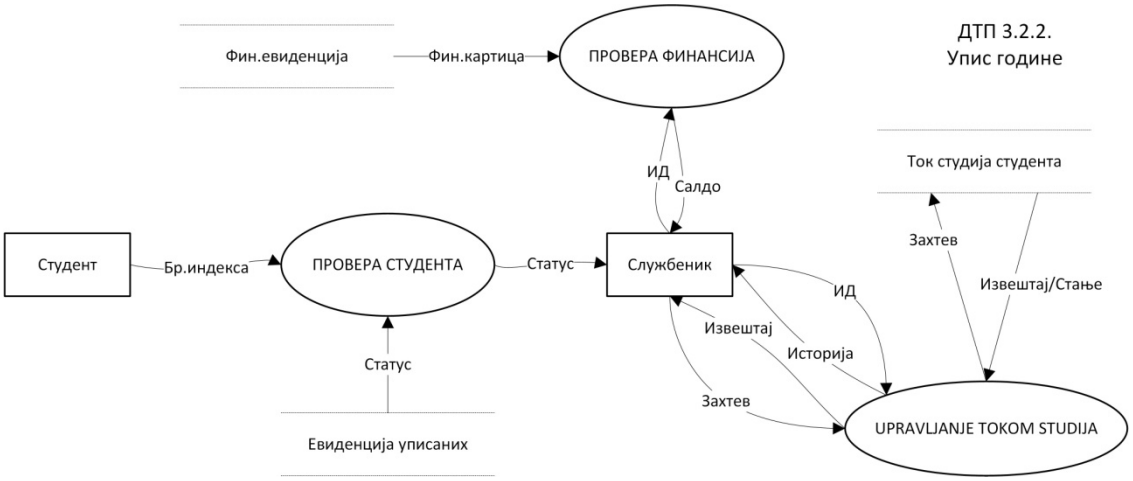
Слика 4: Итерације софтверског процеса према предложеној методологији

Развој софтвера је екстремно сложена активност. Софтверско инжењерство се бави развојем ефикасног и поузданог софтвера на систематичан и контролисан начин користећи формалне методе. Ипак, софтверски производ није материјалан, резултат и даље често зависи од индивидуалних вештина појединаца и способности тимског рада. Софтверско инжењерство се и даље интензивно развија, није достигло своју „зрелу“ фазу.

Ако је одлука о коришћењу методологије донета на почетку развоја новог софтвера, предложена методологија предвиђа коришћење ССА у фазама анализе и дизајна. У оквиру изабраног модела софтверског процеса који ће се користити у развоју, ССА треба користити за спецификацију и моделовање. Ако је у питању софтвер који је у развоју, или софтвер који је већ у фази одржавања, и донета је одлука о коришћењу предложене методологије, предвиђена је анализа примењеног модела развоја и коришћења ССА у софтверском процесу. Уколико у спроведеном развоју није коришћена ССА, треба спровести накнадну анализу. Резултат спроведене ССА треба да буде прецизно дефинисана хијерархија процеса представљена у облику скупа дијаграма тока података.

3.2. Структурна системска анализа

ССА као метода за моделовање и спецификацију информационог система користи се као методолошки поступак декомпозиције система на подсистеме. Процеси обраде података се представљају помоћу дијаграма токова података. Пример дијаграма тока података процеса "Упис године" из [61] приказан је на слици 5.



Слика 5: Пример дијаграма тока података "Упис године"

ССА је једна поптуна, јединствена метода за анализу и функционалну спецификацију информационог система, односно софтвера. Она се на различите начине може повезати са методама других фаза у неку специфичну методологију целокупног развоја ИС-а [62].

Применом методе ССА у оквиру ове методологије предвиђа се добијање одговора на следећа питања:

- Које функције поседује систем или које функције треба да има будући информаиони систем?
- У каквом су међусобном односу те функције?
- Какве трансформације систем треба да врши?
- Која су и каква складишта података потребна за процесе обраде?
- Одакле долазе улазни подаци и какву структуру имају?
- Куда одлазе излазни подаци као резултати обраде?

Главни циљ развоја, односно увођења информационог система је да се информационо подрже суштински процеси који се одвијају у организацији. Посматрајући систем са аспекта његових функција, као резултат примене ССА добија се модел процеса разматраног система. Важна предност ове методе имајући у виду циљеве предложене методологије, је то што у својој основи садржи парадигму протока података. Примена ове методе обезбеђује хијерархијски опис процеса посматраног система. Метода омогућава поступно декомпоновање сложених процеса система на потпроцесе, све до нивоа када више није потребно декомпоновати процесе, до нивоа примитивних процеса. Оваква функционална спецификација система и документовани модели пословних процеса организације, пружају основу за анализу грешака које се јављају у току коришћења система, односно у фази одржавања софтвера посматрано кроз софтверски процес.

Резултат примене ССА у склопу ове методологије, било као интегралног дела софтверског процеса, било као накнадне активности анализе и моделовања, треба да буде потпуна спецификација ИС-а:

- Хијерархијски организован скуп дијаграма тока података
- Речник података свих токова и складишта података
- Модел података , дијаграм ентитета и веза (ЕР дијаграм)
- Спецификација логике свих примитивних процеса

Веома је важна израда процесног модела система, који је основа за анализу грешака. Исправност елемената сваког дијаграма тока података, као и конзистентност дијаграма контекста са дијаграмима нижег нивоа, као и конзистентност било ког дијаграма тока података са дијаграмима нижег нивоа, мора бити проверена. Ова детаљна провера уз сва ограничења људских способности представља озбиљан задатак подложан грешкама. Правила за израду дијаграма тока података су једноставна и могу бити изражена обичним говорним језиком и без формализма. Дефинисање формалних правила међутим омогућава аутоматизовану проверу исправности појединачних дијаграма као и проверу

конзистентности између дијаграма различитог нивоа. То је кључно за функционисање софтверских алата за подршку процесу израде дијаграма тока података. Такви софтверски алати, CASE алати, су предвиђени овом методологијом.

Формални модел правила за дијаграме тока података, представљен математичком нотацијом из [63] је надаље приказан:

Дефиниција 1:
Нека је D дијаграм тока података, тада је
D = {P, F, S, E} , где је

P = $\{p_1, p_2, p_3\ldots, p_m\}$ је коначни скуп процеса;

F= $\{f_1, f_2, f_3\ldots, f_m\}$ је коначни скуп токова података;

S = $\{s_1, s_2, s_3\ldots, s_m\}$ је коначни скуп складишта података;

E = $\{e_1, e_2, e_3\ldots, e_m\}$ је коначни скуп ентитета;

Дефиниција 1 дефинише дијаграм тока података. Дијаграм тока података састоји се од скупа процеса, токова података, складишта података и ентитета.
Дефиниција 2:
Нека је C дијаграм контекста, тада је
C={<ei,fj,p1>,<p1,fk,ei>},
$fj \neq fk, \forall 1 \leq i,j,k \leq m$
Дефиниција 2 дефинише дијаграм контекста. Дијаграм контекста састоји се од тачно једног процеса и скупа ентитета и скупа токова података. Ток података може бити повезан од екстерног ентитета ка процесу и обрнуто, али то морају бити различити токови података. Складиште података не може бити садржано на дијаграму контекста.
Дефиниција 3:
За дато D = {P, F, S, E},
процес p, где $p \in P$, је јединствен, и важи
$\forall pi,pj \in P, pi \neq pj, 1 \leq i,j \leq m$

Из Дефиниције 3, име процеса је јединствено. Иста правила важе и за токове података (Дефиниција 4), складишта података (Дефиниција 5) и ентитете (Дефиниција 6).

Дефиниција 4:

За дато $D = \{P, F, S, E\}$,

ток података f, где $f \in F$ је јединствен, и важи

$\forall fi,fj \in F, fi \neq fj, 1 \leq i,j \leq m$

Дефиниција 5:

За дато $D = \{P, F, S, E\}$,

складиште података $s \in S$ је јединствено, и важи

$\forall si,sj \in S, si \neq sj, 1 \leq i,j \leq m$

Дефиниција 6:

За дато $D = \{P, F, S, E\}$,

ентитет $e \in E$ је јединствен, и важи

$\forall ei,ej \in E, ei \neq ej, 1 \leq i, j \leq m$

Дефиниција 7:

За дато $D = \{P, F, S, E\}$,

и дијаграм контекста C, важи

$\forall fi \in C, \exists fj \in D, fi = fj, 1 \leq i,j \leq m$

Дефиниција 7 указује да сваки ток података који припада дијаграму контекста мора постојати и на дијаграму тока података. Исто правило важи и за ентитете (Дефиниција 8).

Дефиниција 8:

За дато $D = \{P, F, S, E\}$,

и дијаграм контекста C, важи

$\forall ei \in C, \exists ej \in D, ei = ej, 1 \leq i,j \leq m$

Дефиниција 9:

За дато $D = \{P, F, S, E\}$,

и за $\forall pi,si,fj,fk \in D$, важи

$D=\{<pi,fj,si>\}$ и

$D=\{<si,fk,pi>\}$, $fj \neq fk, 1 \leq i,j,k \leq m$

Дефиниција 9 дефинише да за сваки ток података који иде од процеса до складишта података може постојати само различит ток података у смеру од тог складишта података до тог процеса.

Дефиниција 10:

За дато D = {P, F, S, E}, важи

$D=\{<ei,fj,pi>\}$ и

$D=\{<pi,fk,ei>\}$,$fj\neq fk,1\leq i,j,k\leq m$

Дефиниција 10 дефинише да за сваки ток података који иде од ентитета до процеса може постојати само различит ток података у смеру од тог процеса до тог ентитета.

Дефиниција 11:

За дато D = {P, F, S, E}, и за

$\forall pi,pj,fj,fk\in D$, важи

$D=\{<pi,fj,pj>\}$ и

$D=\{<pj,fk,pi>\}$, $fj\neq fk,1\leq i,j,k\leq m$

Дефиниција 11 дефинише да за сваки ток података који иде од једног процеса до другог процеса може постојати само различит ток података у супротном смеру.

Претпоставка 1:

За дато D = {P, F, S, E}, и за

$\forall ei,ej,fk\in D$, важи

$D\neq\{<ei,fk,ej>\},1\leq i,j,k\leq m$

Претпоставка 2:

За дато D = {P, F, S, E}, и за

$\forall si,sj,fk\in D$, важи

$D\neq\{<si,fk,sj>\},1\leq i,j,k\leq m$

Претпоставка 1 указује да не може постојати ток података који повезује два ентитета, а Претпоставка 2 указује да не може постојати ток података који повезује два складишта података.

Претпоставка 3:

За дато D = {P, F, S, E}, и за

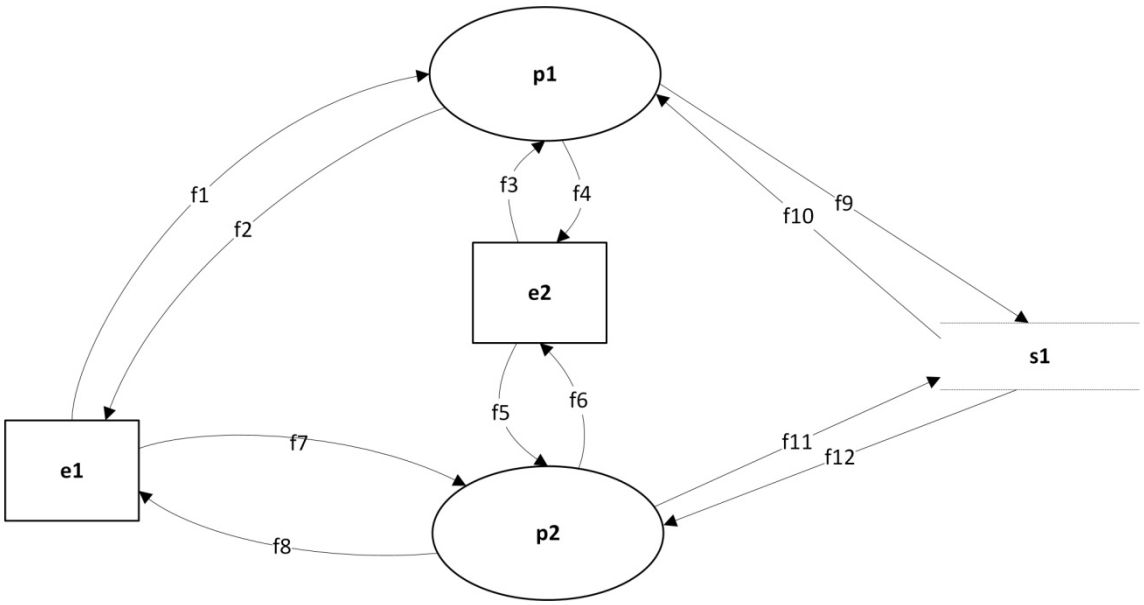
$\forall ei,sj,fk\in D$, важи

D≠{<ei,fk,sj>} и D≠{<sj,fk,ei>},

∀ei,pi,fj,fk∈D, 1≤i,j,k≤m

Претпоставка 3 указује да не може постојати ток података који повезује ентитет и складиште података нити складиште података и ентитет.

Сваки дијаграм се по овом моделу може представити скупом токова података. Сваки ток података се представља као уређена тројка од извора, тока и одредишта. Извор и одредиште могу бити ентитети, складишта или процеси. Први елемент тројке је извор података, а последњи елемент је одредиште података. Пример тока података са графичким и формалним записом приказан је на слици 6.

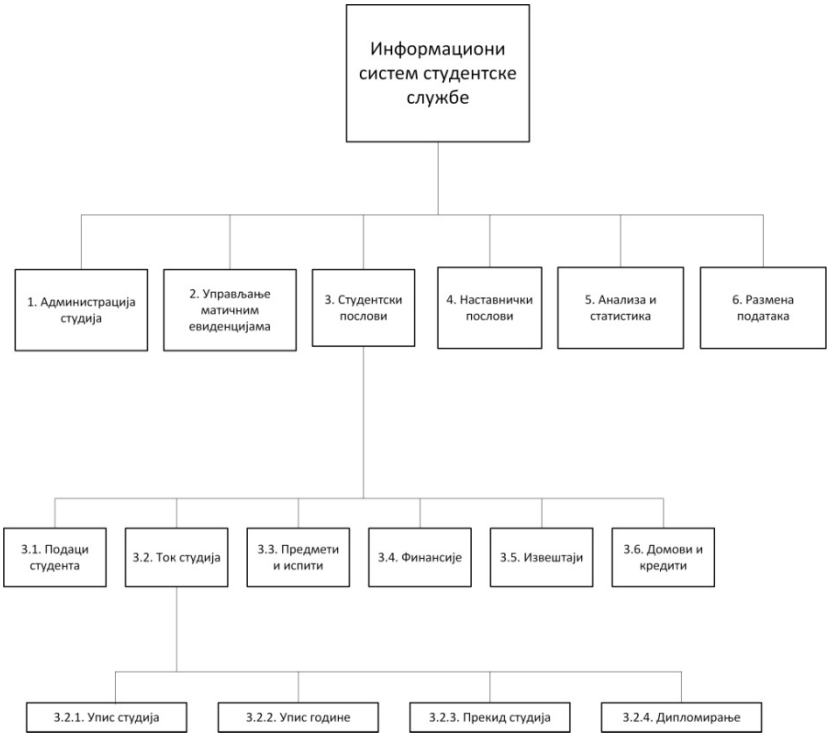


Слика 6: Дијаграм тока података, графички и формални запис

Формални запис дијаграма са слике 6: D = { <e1,f1,p1> , <p1,f2,e1> , <e2,f3,p1> , <p1,f4,e2> , <e2,f5,p2> , <p2,f6,e2> , <e1,f7,p2> , <p2,f8,e1> , <p1,f9,s1> , <s1,f10,p1> , <p2,f11,s1> , <s1,f12,p2>}

3.3. Ефекти протока података

Информациони систем може бити веома сложен. Дијаграм тока података који представља графички модел било ког система без обзира на сложеност система користи само четири основна симбола. То су симболи који представљају: процесе обраде – функције система, складишта података, ентитете-објекте окружења или интерфејсе, и токове података. Да би се обезбедио јасан и одговарајуће детаљан опис система, користи се метода апстракције. Као што је предвиђено ССА методом, треба формирати почетни дијаграм на највишем нивоу апстакције, а потом декомпоновањем процеса вишег нивоа на детаљније процесе нижег нивоа апстракције спровести поступак хијерархијске декомпозиције. На слици 7, приказан је пример дела дијаграма декомпозиције.



Слика 7: Део дијаграма декомпозиције из ССА информационог система студентске службе Машинског факултета

За процесе који се не декомпонују - примитивне процесе, даје се опис логике функционисања. Уобичајено је спецификацију примитивних процеса

представити у облику псеудокода или дијаграма тока. Логика рада примитивних процеса треба да буде релативно једноставна, да може да се приближно представи на једној страници. Паралелизам међу процесима је карактеристика одвијања процеса у организацији. Дијаграм тока података је начин представљања у основи паралелних процеса. Дијаграм тока података не говори ништа о редоследу извршавања процеса. Када се дође до процеса који се могу представити секвенцијално, и који ће се одвијати као рачунарска обрада, тада се користи други начин описа процеса погодан за приказ секвенцијалног тока. Као и процесе, потребно је декомпоновати и токове података и складишта података ако су у питању уније података. Информације о декомпозицији података се налазе у Речнику података, и садрже опис структуре података.

Примитивни процеси покривају цео простор система. Можемо их замислити као елементе којима је покривен цео простор система. Процеси вишег нивоа су у таквом моделу само блокови елемената и блокова, те као такви могу бити корисни на вишем нивоу апстракције. За потребе анализе резултата, ефеката обраде и протока података у систему, у овом докторском раду анализирају се примитивни процеси. За разлику од моделирања система путем хијерархијске декомпозиције, када се користи "од горе на доле" метода (top-down), анализа ефеката протока података заснована на праћењу индикатора резултата примитивних процеса, спроводи се супротним смером, од доле на горе. Моделирање процеса, као и свако друго моделирање је високо креативна делатност. Резултат зависи од стручних квалитета и искуства особа које релизују анализу. Не могу се дати прецизне и комплетне инструкције, детаљно и са корак по корак упутствима за моделирање. Суштински паралелни процеси међусобно комуницирају преко меморије система. Меморија система, у општем случају су складишта података. Они су основа и услов за паралелизам процеса. Анализом сваког примитвног процеса треба установити шта је резултат обраде података и у ком складишту се чувају резултати обраде.

Перформанса процеса је учинак или успех процеса. Перформанса мора бити изражена неком величином. Мерење перформанси је утврђивање вредности перформансе, односно, мерење вредности учињеног. Мере перформанси су

нумерички или квантитативни индикатори који показују колико се добро остварује сваки циљ [64].

Индикатори су квантитативни или квалитативни показатељи помоћу којих се, директно или индиректно, процењује или мери ниво остварења одређеног циља, брзина, односно утрошено време остварења циља (Business Process management - Pocket Guide, n.d.).

У [65] су дефинисали пет основних критеријума које индикатори перформанси процеса треба да задовоље, а то је да су:

- тачни
- лако разумљиви
- правовремени
- оријентисани на акцију
- са имплементацијом која није скупа

Праћење, мерење и чување вредности индикатора перформанси процеса временски је дефинисано. Прикупљени подаци из фазе одржавања различитих верзија софтверског решења, могу се међусобно поредити. Резултати поређења могу да се користе као основа за одлучивање о променама у софтверском решењу. Не треба мерити успешност процеса користећи непотребан број индикатора, треба се фокусирати на индикаторе перформанси, на оне који ће стварно пратити ефикасност процеса.

Према [66], индикатори могу бити:

- Иницијални индикатори или индикатори структуре, ови индикатори се користе за праћење ресурса процеса, то могу бити: производни погони, људи, новчана средства, остварене услуге и слично;
- Прелазни индикатори или индикатори одвијања процеса, ови индикатори пружају информације о стању процеса;

- Финални индикатори или индикатори резултата, ови индикатори одговарају на питања о излазима процеса и да ли су ефекти процеса они очекивани. То су најзначајнији индикатори који омогућавају оцену коначних резултата процеса. Овај тип индикатора се користи у оквиру предложене методологије како би се квантификовалии ефекти протока података посматраних процеса.

Користећи претходно формирани модел процеса, за сваки примитивни процес ис ССА користећи псеудокод процеса и речник података, одредити јединствене ефекте сваког процеса обраде у оквиру примитивног процеса. Сваки појединачни ефекат треба да буде прецизно дефинисан променом на подацима у складишту података који је део процеса и приказан је на дијаграму тока података. У фази логичког дизајна решења, идентификовати ефекте и одредити индикаторе који нумерички одређују те ефекте.

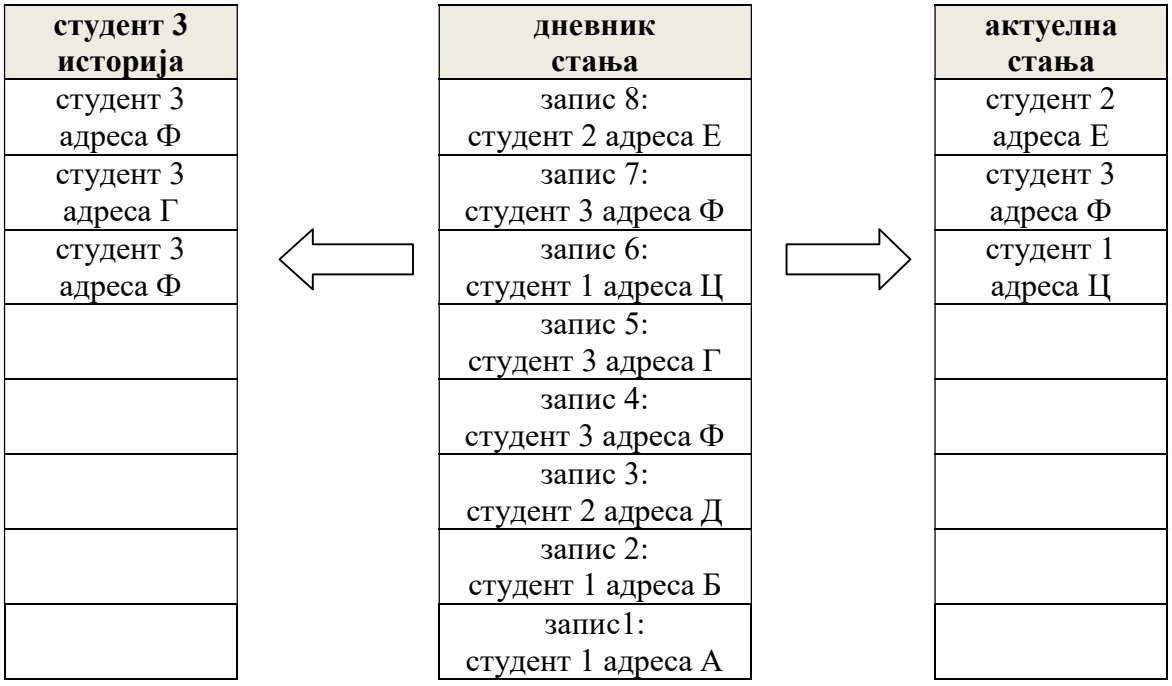
Постоји много теоријских и из праксе потврђених модела за мерење перформанси. Сваки од ових модела успоставља мерење перформанси из различитих перспектива. Понуђене су бројне дефиниције система за мерење перформанси, али још увек не постоји усаглашен договор које су то основне компоненте и карактеристике система [67]. У оквиру предложене методологије предвиђа се систем мерења перформанси делом по угледу и са елементима DOE/NV модела који је предложио U.S. Department of Energy Nevada Operations Office [68]. DOE/NV методологија предвиђа приступ који је оријентисан на процесе, и ослања се на идеју континуалног побољшања система, и мада није прецизна у опису како идентификовати процесе и како одредити које мере перформанси треба пратити, компатибилна је са елементима методологије. Већина познатих модела само даје назнаке како дефинисати индикаторе перформанси.

3. 4. Надгледања процеса система

Резултат обраде података у процесу, може да буде сачуван у складишту података, али може и да буде само прослеђен екстерном кориснику, интерфејсу. Посматрано на дијаграму тока података то је питање смера тока података. Како је један од елемената ове методологије, евидентирање и трајно чување информација о свим реализованим обрадама података, потребно је евидентирати и процесе који природно не захтевају евидентирање. Разни извештаји, анализе, потврде, уверења и слично, треба да се евидентирају у складишту података, са својим суштинским и пратећим подацима који описују конкретну активност. Разлог за овакав приступ, није само потреба за праћењем индикатора, већ и пословне и организационе природе, и ако није садржан у нефункционалним захтевима дефинисаним од стране стејкхолдера, ипак, у предложеној методологији треба да се примени доследно. Најважнији организациони разлози за овакав приступ су потреба за евиденцијом извештаја који су излаз из информационог система и могућност поновног накнадног издавања истог извештаја независно од промењеног скупа изворних података.

Процеси у оквиру разматраног пословног информационог система, које посматрамо кроз парадигму протока података, могу да генеришу различите структуре података. Некада су те структуре по природи у облику дневника догађаја. Модел података у облику дневника догађаја или стања је начин како треба организовати евидентиране податке за све процесе обраде које треба пратити. Овакав модел података примењен чак и када то није природни облик тих података пружа основу за жељено праћење процеса обраде. То значи, да се ако је у питању дневник стања, не чува само један скуп података (запис) који приказује актуелно стање. У дневнику, најчешће је то у имплементацији једна табела, чувају се сва претходна и тренутно стање неког ентитета. Одређивање тренутног стања, своди се на идентификовање последњег записа за тај ентитет. Недостаци оваквог приступа су: повећање потребног меморијског простора за чување података и спорији рад. Предности су: доступност информација о свим претходним стањима - историја стања, могућност анализе претходних промена и слично. У овом

приступу, брисање података није класично брисање, већ замена актуелног стања новим стањем које може представљати статус обрисаног стања. На слици 8 приказан је пример оваквог приступа на евиденцију адресе студента. Дневник адреса садржи основне детаљне податке, а информацију о актуелним адресама студената добијамо као извештај из основне табеле. Историју промена адреса такође можемо да добијемо као извештај, изведене податке, као специфичан поглед на податке.



Слика 8: Пример модела "дневник стања"

Физичка имплементација надгледања процеса у оквиру ове методологије може бити реализована суштински на два начина. Један начин је коришћење интерних механизама система базе података, а други је коришћење апликативног слоја решења. Могућа је и комбинација метода у зависности од начина имплементације посматраног система, коришћених технологија и сложености конкретног процеса који се посматра.

Надгледање система коришћењем интерних механизама система базе података зависи од особина и софтверских алата конкретног RDBMS-а. Овај приступ има следеће предности: контрола обраде података је логички близу самим подацима, имплементација је независна од апликативног слоја решења, једноставније је остварити аутоматизам и аутономност у односу на основни софтвер, није потребно додатно управљање у ситуацији када више различитих софтвера користи исте податке што може бити одлучујући фактор.

Апликативни приступ надгледању система може да се реализује уградњом потребних функциопналности у постојећи софтвер, или развојем нове засебне специјализоване апликације. Доступност изворног кода и документације основне апликације је важан предуслов за овај приступ. Предност оваквог приступа је релативна независност у односу на конкретан RDBMS. Овакав приступ континуалном надзору предлажу у [69].

Модерни релациони системи база података имају могућност аутоматског вођења дневника трансакција базе података. Проблем може представљати то што су записи у овим дневницима често недовољно документовани и веома сложени, па су стога тешки за примену и анализу. Користан су механизам за форензику базе података. Једноставнији метод документовања активности над подацима у релационим базама података може бити коришћење окидача базе података. Окидачи (енгл. Trigers) представљају посебну врсту програмских процедура које се аутоматски извршавају у бази података као одговор на дефинисани догађај. Тај догађај је уобичајено нека операција над подацима у одређеној табели. Код окидача имамо три битне компоненте, то су догађај, услов и акција. Догађај може бити упис неког података, ажурирање или брисање податка, услов дефинише додатне провере које одређују да ли ће се покренути окидач и спровести предвиђена аутоматска акција.

У основи све операције са подацима у складишту података могу се свести на основне четири: читање, писање, измена и брисање. Позната је CRUD скраћеница из софтверске терминологије, која значи: create, read, update i delete. Овај израз је постао популаран помињањем у "Managing the Database Environment" [70]. Овај акроним не мора да се односи само на команде за управљање подацима

у релационим базама података, може да се примени и на друге језике и протоколе, приказано у табели 3.

Табела 3: CRUД

Operacija	**SQL**	**HTTP**	**REST servisi**	**DDS**
Create	INSERT	PUT / POST	POST	write
Read (Retrieve)	SELECT	GET	GET	read / take
Update (Modify)	UPDATE	PUT / POST / PATCH	PUT	write
Delete (Destroy)	DELETE	DELETE	DELETE	dispose

Окидачи се лако имплементирају и флексибилни су, могу се лако модификовати и повезивати у сложеније процедуре. За потребе евидентирања обрада података најчешће се користе окидачи који се извршавају после спроведене обраде (after triggers). Треба приметити да у већини RDBMS-ова нема окидача за команду читања података (select) што је познат захтев програмерске заједнице [14].

Често се за потребе праћења индикатора користе OLAP (On-Line Analytical Processing) складишта података [71]. OLAP системи претварају класичне трансакционе оперативне колекције података у облик погодан за аналитичке обраде. Класични трансакциони системи OLTP (Operational Transaction System) намењени су за обраду трансакција над подацима и обично садрже једноставне CRUD операције. OLAP системи служе за испитивање сложених веза великих колекција података. У табели 4 приказано је поређење OLTP и OLAP система.

Табела 4: Поређење OLTP и OLAP система - Turban [72]

	OLTP	**OLAP**
Сврха	Свакодневне пословне фунцкије	Подршка одлучивању и одговори на упите менаџмента
Подаци	Трансакциона база података са	Складиште података са

	нормализованим подацима уз фокус на ефикасност и конзистентност	денормализовани подацима уз примарни фокус на тачност и комплетност
Извештаји	Рутински, периодични, уско фокусирани извештаји	Ad hoc, мултидимензионална анализа, извештаји широког фокуса
Ресурси	Релационе базе података	Велики капацитет и специјализоване базе података, потребни вишепроцесорски сервери са великим RAM-ом
Брзина рада	Брза обрада пословних трансакција и извештаја	Спори сложени упити над великим количинама података

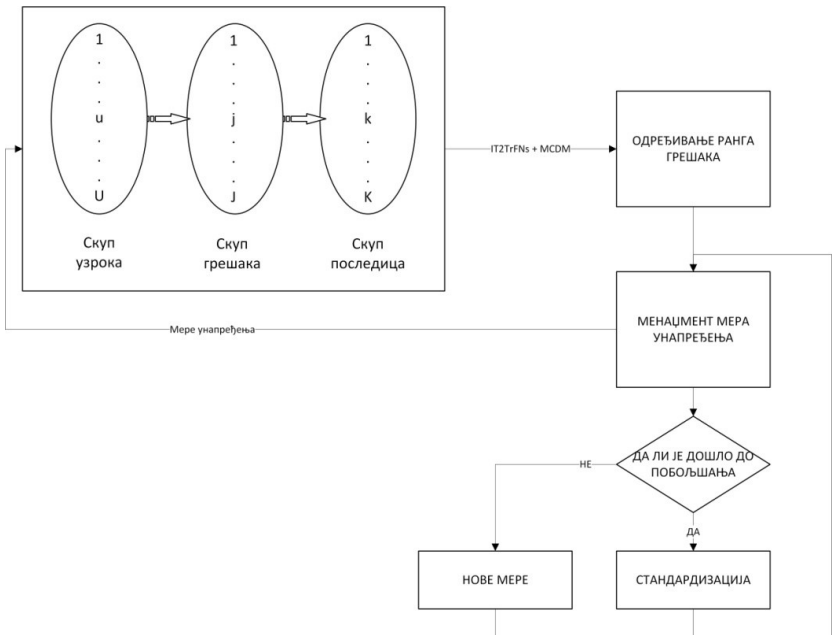
Респектујући велике захтеве за реусрсима које имају OLAP системи и чињенице да OLTP системи раде на нивоу појединачних трансакција и да користе релационе базе података које су по правилу дизајниране у складу са апликацијама које их користе, у овом истраживању је примењен приступ апликативног управљања подацима за дефинисане индикаторе уз коришћење релационе базе података.

У фази дизајна, као и у фази имплементације дизајна софтвера (у току програмирања кода и креирања базе података) овом методологијом се предлаже приступ доследног евидентирања свих процеса који користе податке било за упис било за читање. Основа за реализацију евидентирања треба да буде у проширеном моделу података који то обезбеђује, и треба да буде имплементирана на најбољи начин у зависности од конкретне технологије примењене за базу података система. На тај начин се обезбеђује основа за праћење ефеката протока података и грешака у раду софтвера.

3.5. Нови хибридни приступ у анализи ризика у фази одржавања софтвера

По завршетку развоја актуелне верзије софтвера у току којег су решени постављени функционални захтеви, и испоруке и почетка коришћења код корисника, прелази се у фазу одржавања софтвера у контексту софтверског процеса. Као основа за анализу, користи се хијерархија процеса формирана у склопу ССА. У току ове фазе, систем је константно праћен како кроз уграђене системске контроле, тако и кроз документацију коју уређују корисници и администратори система.

Анализа ризика било које ставке система, тако и софтвера може да буде заснована на правилима која су дата у FMEA анализи [22]. Може да се каже да се ова анализа реализује кроз неколико корака: (1) идентификовање грешака, (2) идентификовање последица које могу да настану услед реализације грешака, (3) анализа узрока који доводе до настајања грешака, (4) рангирање грешака, (5) предузимање мера које треба да доведу до смањивања или редуковања фактора ризика и (6) оцена ефеката предузетих мера. Поступак анализе грешака је приказан на слици 9 и надаље детаљно описан.



Слика 9: Поступак анализе грешака

3.5.1. Идентификовање грешака софтвера

Идентификовање потенцијалних грешака софтвера у посматраним процесима може да се врши применом квалитативних метода одлучивања као на пример: браинсторминг сесија, округли сто, Делфи метода итд. Другим речима, проблем идентификовање грешака софтвера се може поставити као проблем групног одлучивања. Доносиоци одлуке треба да буду инжењери који учествују у развоју. Они своје процене доносе на основу дневника-логова који су како системски тако и ручно вођени. Може да се претпостави да они одлуку доносе консензусом.

Формално, све идентификоване грешке се представљају скупом индекса $\iota = \{1,...,j,...,J\}$. Укупан број идентификованих грешака је означен као J. Индекс j, j=1,..,J означава идентификовани грешку.

3.5.2. Идентификовање последица грешака софтвера

Свака идентификована грешка j, j=1,..,J може да доведе од настајања једне или више последица које се формално представља скупом $\lambda = \{1,...,k,...,K\}$. Укупан број последица је означен као К и к, к=1,..,К је ознака за идентификовану последицу. Скуп последица одређују доносиоци одлуке на основу искуства. Ако грешка j, j=1,..,J доводи настајања две или више последица, према FMEA методи, неопходно је да се посебно анализира сваки уређени пар (грешка, последица).

3.5.3. Идентификовање узрока грешака софтвера

Материјализација једног или више узрока доводи до материјализације сваке идентификоване грешке. Сви узроци могу да се класификује у више група према области ИС. Тако узрок грешке може да се налази у софтверу информационог система, апликативном или системском. Може да буде хардверске природе, узрок може да буде људски фактор или у дефинисаним

процедурама организације. Узрок може да буде и комплекснији и да обухвата више различитих области. Узроци на које је ова методологија посебно усмерена и које треба да најуспешније елиминише јесу узроци из апликативног софтвера. Тесна интеграција FMEA анализе - анализе ризика са једне стране и модификованог итеративног софтверског процеса, управо то треба да омогући.

У овом докторском раду сматра се да свака врста узрока има исту важност за функционисање софтвера. Респектујући ову претпоставку, могуће је да се формира јединствена листа узрока. Формално сви узроци могу да се представе скупом индекса $\lambda = \{1,\ldots, u,\ldots, U\}$.

Одређивање узрока је засновано на знању и искуству доносилаца одлука као и резултатима најбоље праксе.

3.5.4. Оцена и рангирање грешака софтвера

Свака грешка софтвера треба да се оцени према више фактора ризика (РФ-а) који се формално записују као скуп индекса критеријума $\kappa = \{1,\ldots, k,\ldots, K\}$. Укупан број разматраних РФ-а је означен као К. Индекс сваког РФ је означен као к, к=1,...,К. У овом докторском раду, идентификовање РФ-а је заснована на FMEA оквиру [22, 30]. Ови РФ-а су: озбиљности последице која настаје услед реализације разматране грешке (к=1), учесталости настајања разматране грешке (к=2) и могућности откривања разматране грешке (к=3).

Неизвесноти у релативној важности разматраних РФ-а и вредности РФ-а на нивоу сваке идентификоване грешке су описане одговарајућим лингвистичким исказима. Развијене су многе области математике, као што је теорија фази скупова [37, 38] којом се на довољно добар начин могу квантитативно описати лингвистички термини.

Када се разматра проблем моделовања неизвесности прво питање се односи на избор броја лингвистичких термина којима се описују разматране неизвесности. Углавном, број неизвесности одређују доносиоци одлука на основу свог знања и искуства респектујући величину проблема. Неки аутори сматрају да

човек може највише истовремено да разматра највише седам различитих ставки тако да се сматра да свака неизвесност може да се опише помоћу највише седам лингвистичких термина.

Друго питање је избор функције расподеле могућности фази бројева којима се квантитативно описују лингвистички искази. У литератури, могу да се нађу многи радови у којима се различите врсте неизвесности и непрецизности моделирају тип-1 фази скуповима. Најчешће се користе троугаони и трапезоидни фази бројеви којима се на довољно добар начин описују неизвесности а истовремено обим рачунања је значајно мањи него ако се користе фази бројеви вишег реда.

Треће питање је дефинисање домена фази бројева. Треба нагласити да не постоје правила нити препоруке како изабрати домен на којима су дефинисани фази бројеви. Најчешће се користи стандардна скала мера [73] која је дефинисана на скупу реалних бројева на интервалу [1-9]. Многи аутори су тип-1 фази бројеве дефинисали на интервалу [1-5], интервалу [0-1] и др.

При изучавању неизвесности многи аутори су сматрали да тип-1 фази број не могу довољно добро да опишу неизвесност. Стога, Zadeh (1975) [43] је увео тип-2 фази скуп који представља проширење тип-1 фази броја. Тип-2 су описани помоћу две функције расподеле могућности тако да је омогућен већи степен слободе и флексибилности при описивању неизвесноти. Респектујући ову особину тип-2 фази броја, може да се каже да ова врста фази броја боље описује неизвесност него тип-1 фази број. Тип-2 фази бројеви захтевају сложенији обим рачунања тако да је њихова применљивост мања него тип-1 фази бројева. Mendel [44] је увео концепт интервалног тип-2 фази бројева. Код ових фази бројева горња и доња функција расподеле могућности има исту вредност за модалне вредности у домену. Многи истраживачи су овај тип фази бројева користили за моделовање неизвесности и непрецизности које егзистирају у разчичитим управљачким проблемима [74, 75].

3.5.4.1. Моделирање релативне важности РФ-а

Сви РФ-а према којима се оцењују идентификоване грешке које могу да доведу до смањења количине и брзине тока података немају исту релативну важност. Многи аутори сматрају да доносиоци одлука могу значајно боље да искажу своје процене релативне важности РФ-а ако користе лингвистичке исказе него прецизне бројеве [52, 76, 77]. Ови лингвистички искази су моделирани трапезоидним фази бројевима типа 2 (IT2TrFN).

Релативна важност РФ-а је задата матрично. Елементи ове фази матрице парова упоређења релативне важности РФ-а су описане унапред дефинисаним лингвистичким исказима. Број и врсту лингвистичких исказа одређују доносиоци одлука. У овом докторском раду се сматра да релативна важност РФ-а на одговарајући начин може да се опише помоћу три лингвистичка термина који су моделирани IT2TrFN:

мала важност (Л)-$\big((1{,}1{,}2{,}5;1),(1{,}1{,}2{,}4;0.75)\big)$
средња важност (М)-$\big((1.5{,}2.5{,}3.5{,}4.5;1),(2{,}2.5{,}3.5{,}4;0.75)\big)$
висока важност (Х)-$\big((1{,}4{,}5{,}5;1),(2{,}4{,}5{,}5;0.75)\big)$

Домени ових IT2TrFN су дефинисани на скупу реалних бројева који се налазе на интервалу [1-5]. Вредност 1 означава да је релативна важност РФ k према РФ k' једнака. Вредност 5 означава да РФ k има екстремно већу важност у односу на РФ k'.
$k, k' = 1,..,K.$

Вредности елемената ове фази матрице парова упоређења релативне важности критеријума се формално представљају IT2TrFN-а:

$$\tilde{\tilde{W}}_{k\,k'} = \Big(\big(a^{U}_{kk'}, b^{U}_{kk'}, c^{U}_{kk'}, d^{U}_{kk'}; \alpha_1, \alpha_2\big), \big(a^{L}_{kk'}, b^{L}_{kk'}, c^{L}_{kk'}, d^{L}_{kk'}; \beta_1, \beta_2\big)\Big)$$

Најмања и највећа вредност у домену ових IT2TrFN-а су означена као $a^{U}_{\text{кк}'}, d^{U}_{\text{кк}'}$, респективно, и $a^{L}_{\text{кк}'}, d^{L}_{\text{кк}'}$, респективно. Модалне вредности су $b^{U}_{\text{кк}'}, c^{U}_{\text{кк}'}$, респективно, и $b^{L}_{\text{кк}'}, c^{L}_{\text{кк}'}$,респективно.

Вредности функције расподеле могућности ових IT2TrFN-а су дефинисане на следећи начин:
$(\alpha_1, \alpha_2) \in [0,1]$означава вредности фунције расподеле могућности за вредности у домену $b^{U}_{kk'}$,$c^{U}_{kk'}$ респективно,
$(\beta_1, \beta_2) \in [0,1]$ означава вредности фунције расподеле могућности за вредности у домену $b^{L}_{kk'}$,$c^{L}_{kk'}$ респективно.

Претпоставимо да РФ k' има већу релативну важност од РФ к, $k, k' = 1,..,K$. У овом случају вредност у фази матрици парова упоређења релативне важности РФ-а се описује IT2TrFN:

$$\tilde{\tilde{W}}_{k\,k'} = \left(\tilde{\tilde{W}}_{k\,k'}\right)^{-1}$$

$$= \left(\left(\frac{1}{d^{U}_{k'\,k}}, \frac{1}{c^{U}_{k'\,k}}, \frac{1}{b^{U}_{k'\,k}}, \frac{1}{a^{U}_{k'\,k}}; \alpha_1, \alpha_2\right), \left(\frac{1}{d^{L}_{k'\,k}}, \frac{1}{c^{L}_{k'\,k}}, \frac{1}{b^{L}_{k'\,k}}, \frac{1}{a^{L}_{k'\,k}}; \beta_1, \beta_2\right)\right)$$

Ако РФ-и к и k' $(k, k' = 1,..,K)$ имају једнаку релативну важност тада се вредност елемента у фази матрици парова упоређења представља прецизним бројем 1. Овај прецизни број може да се запише IT2TrFN $\left((1,1,1,1; \alpha_1, \alpha_2), (1,1,1,1; \beta_1, \beta_2)\right)$.

3.5.4.2. Израчунавање вектора тежине РФ-а

У овом докторском раду релативна важност разматраних РФ-а је задата помоћу фази матрице парова упоређења, $\left[\tilde{\tilde{W}}_{k\,k'}\right]_{KxK}$. Елементи ове матрице су дефинисани као релативна важност РФ к према РФ $k, k' = 1,...,K; k \neq k'$ који су процењени од стране доносиоца одлука. Може се сматрати да доносиоци одлука одлуку доносе консензусом.

Доносиоци одлуке могу да направе грешке када врше процену релативне важности РФ-а. Неопходно је да се утврди колико пуно те грешке утичу на тачност одређивања тежине РФ-а. Провера конзистентности може да се изврши применом различитих метода које су развијене у литератури. Најшире коришћена метода је метода сопственог вектора [73] која према мишљењима многих аутора представља природну меру неконзистентности.

Да би могла да се примени ова метода неопходно је да фази матрица парова упоређења релативне важности РФ-а, $\left[\tilde{\tilde{W}}_{k\,k'}\right]_{KxK}$ буде трансформисана у матрицу парова упоређења релативних важности РФ, $[W_{k\,k'}]_{KxK}$.Елементи матрице$[W_{k\,k'}]_{KxK}$, $W_{k\,k'}$, $k, k' = 1, \ldots, K;\ k \neq k'$су описане прецизним бројевима који су дефинисани на интервалу као и домени IT2TrFN. Репрезентативни скалари, $W_{k\,k'}$ дефинисаних IT2TrFN су израчунати применом методе дефазификације која је предложена у [52]:

$$DTraT\,(\tilde{\tilde{W}}_{k\,k'}) = W_{k\,k'}$$

$$= \frac{1}{2} \cdot \left(\begin{array}{l} \dfrac{\left(d^{U}_{k'\,k} - a^{U}_{k'\,k}\right) + \left(\alpha_2 \cdot b^{U}_{k'\,k} - a^{U}_{k'\,k}\right) + \left(\alpha_1 \cdot c^{U}_{k'\,k} - a^{U}_{k'\,k}\right)}{4} + a^{U}_{k'\,k} + \\ \dfrac{\left(d^{L}_{k'\,k} - a^{L}_{k'\,k}\right) + \left(\beta_2 \cdot b^{L}_{k'\,k} - a^{L}_{k'\,k}\right) + \left(\beta_1 \cdot c^{L}_{k'\,k} - a^{L}_{k'\,k}\right)}{4} + a^{L}_{k'\,k} \end{array} \right)$$

Поступак провере конзистентности применом методе сопственог вектора надаље је изложен.

Потребан услов да матрица парова упоређења релативне важности критеријума, $[W_{k\,k'}]_{KxK}$буде конзистента је да су све вредности $W_{k\,k'}$ позитивне и реципрочне у односну на главну дијагоналу. Нека је овај услов задовољен. Довољан услов да матрица $[W_{k\,k'}]_{KxK}$ буде конзистентна је да коефицијент конзистентности буде мањи или једнак 0,1.

Главна сопствена вредност се израчунава из матричне једначине:

$$[W_{kk'}] \cdot w = \lambda_{\max} \cdot w$$

где је:

w вектор тежине

$\lambda_{\max}$ највећа вредност сопственог вектора матрице $[W_{k\,k'}]_{KxK}$

Ако и само ако је матрица $[W_{k\,k'}]_{KxK}$ конзистентна тада је $\lambda_{\max}$=К.

Мера неконзистентности се рачуна према изразу:

$\lambda_{\max}$- К

Однос конзистенције, C.R. се рачуна према изразу:

$$C.R. = \frac{\lambda_{\max} - K}{K - 1}$$

Сопствене вредности, P.I. зависе од димензије матрице и приказане су у Табели 5.

Табела 5: Вредности P.I. у зависности од димензија матрице

К	3	4	5	6	7	8	9	10	11	12	13	14
P.I.	0.58	0.90	1.12	1.24	1.32	1.41	1.45	1.49	1.51	1.48	1.56	1.57

Индекс конзистенције (C.I.) се рачуна према изразу:

C.I.=C.R./R.I.

Сматра се да је матрица парова упоређења релативних важности РФ-а конзистентна само ако је $C.I. \leq 0.01$. Ако није задовољен овај услов тада се неопходно да доносиоци одлуке врше поново процену.

Вектор тежина РФ-а, $\left[\tilde{\tilde{w}}_k\right]_{1xK}$ се рачуна према процедури која је развијена у [52], тако да:

$$\tilde{\tilde{w}}_k = \left((a_k^U, b_k^U, c_k^U, d_k^U;\ \alpha_1, \alpha_2), \left((a_k^L, b_k^L, c_k^L, d_k^L;\ \beta_1, \beta_2)\right)\right)$$

где:

$$a_k^U = \frac{\sqrt[K]{\prod_{k=1,..,K}\left(a_{k\,k'}^U\right)}}{\sum_{k=1}^{K}\sqrt[K]{\prod_{k=1,..,K}\left(a_{k\,k'}^U\right)}}, b_k^U = \frac{\sqrt[K]{\prod_{k=1,..,K}\left(b_{k\,k'}^U\right)}}{\sum_{k=1}^{K}\sqrt[K]{\prod_{k=1,..,K}\left(c_{k\,k'}^U\right)}},$$

$$c_k^U = \frac{\sqrt[K]{\prod_{k=1,..,K}\left(c_{k\,k'}^U\right)}}{\sum_{k=1}^{K}\sqrt[K]{\prod_{k=1,..,K}\left(b_{k\,k'}^U\right)}}, \; d_k^U = \frac{\sqrt[K]{\prod_{k=1,..,K}\left(d_{k\,k'}^U\right)}}{\sum_{k=1}^{K}\sqrt[K]{\prod_{k=1,..,K}\left(a_{k\,k'}^U\right)}}$$

3.5.4.3. Моделирање вредности РФ-а идентификованих грешака

У овој секцији приказан је начин моделовања озбиљности последице које настаје услед реализације идентификованих грешака, вероватноће настајања грешака и могућност откривања грешака. Доносиоци одлука процењују вредности грешака са сваког разматраног аспекта. Они своје процене заснивају на: (1) карактеристикама сличних софтверских решења или софтверским контролама за аутоматску проверу, и (2) проценама од стране самих корисника, тамо где су корисници активни у обради података.

Озбиљност сваке консеквенце која настаје услед материјализације сваке грешке може да се оцени са различитих аспеката. У конвенционалној FMEA методи, озбиљност последица се оцењује само са апекта квалитета. У [24] су увели претпоставку да озбиљност последице треба да се процени како са аспекта квалитета тако и са аспекта трошкова и сигурности. Како се у овом докторском раду разматра проблем оцене грешака у софтверу може се сматрати да консеквенце које настају услед реализације грешака софтвера има смисла оцељивати само са аспекта тока података. Озбиљност консеквенци су процењене од стране доносилаца одлука који користе седам унапред дефинисаних лингвистичких исказа који су моделирани IT2TrFN као што је приказано у Табели 1. Домени ових фази бројева су дефинисани на скупу реалних бројева који припадају интервалу [0-1]. Вредност 0, односно вредност 1, респективно, означава да разматрана консеквенца апсолутно нема, односно апсолутно има утицај на ток података, респективно.

Учесталост настајања грешке може да се одреди на различите начине. У конвенционалној FMEA, која је првенствено развијена за примену у ауто индустрији, ове вредности могу на довољно добар начин да се одреде на основу података из евиденције. У другим гранама индустрије, подаци из евиденције готово да не постоје или ако постоје не може се сматрати да постоји довољно велики број за примену статистичке анализе ових података. У домену развоја софтвера може да се каже да подаци из евиденције готово и не постоје иако постоје није могуће да се тако једноставно искористе за нови софтвер. Стога, у овом докторском раду, учесталост настајања грешака се одређује на основу процене доносилаца одлука. Дефинисани су пет лингвистичких исказа којима може на довољно добар начин да се опише учесталост настајања грешака. Ови лингвистички искази су моделирани IT2TrFN. Домени ових фази бројева су дефинисани на скупу реалних бројева који припадају интервалу [0-1]. Вредност 0, односно вредност 1, респективно, означава да се грешка не материјализује, односно да се грешка сасвим сигурно материјализује, респективно.

Могућност откривања грешака зависи да ли постоји развијена процедура и да ли се та процедура примењује. Лингвистички искази који су моделирани IT2TrFN су приказани у Табели 1. Домени ових IT2TrFN су дефинисани на интервалу [0-1]. Вредност 0, односно 1 означава да апсолутно не постоји, односно постоји, могућност да се грешка открије, респективно.

Дефинисани лингвистички искази као и одговарајући IT2TrFN су приказани у Табели 6 и Табели 7.

Табела 6: Озбиљност последица и могућност детекције грешака

Лингвистички искази	*IT2TrFNs*	*Опис за озбиљност последица*	*Опис за могућност детекције*
Без последица на проток података / Није могуће детктовати (СД1)	$\left(\begin{matrix}(0,0,0.2,0.3;1,1)\\,(0,0,0.2,0.25;0.8,0.8)\end{matrix}\right)$	*Неприметне или немерљиве последице*	*Апсолутно није могуће детектовати грешку*
Врло мали утицај на	$\left(\begin{matrix}(0.1,0.2,0.3,0.4;1,1)\\,(0.15,0.2,0.3,0.35;0.8,0.8)\end{matrix}\right)$	*Мало успорење*	*Грешку готово није могуће*

проток података / Готово немогуће детектовати (СД2)		*мање битног протока података*	*детектовати*
Мали утицај на проток података / Врло мала могућност детекције (СД3)	$\left(\begin{matrix}(0.2,0.3,0.4,0.5;\ 1,1)\\ ,(0.25,\ 0.3,0.4,0.45;\ 0.8,0.8)\end{matrix}\right)$	*Мало успорење протока података или прекид протока необавезних података процеса*	*Грешка се може детектовати од стране обученог специјалисте*
Средњи утицај на проток података / Средње мала могућност детекције (СД4)	$\left(\begin{matrix}(0.3,0.4,0.5,0.6;\ 1,1)\\ ,(0.35,\ 0.4,0.5,0.55;\ 0.8,0.8)\end{matrix}\right)$	*Привремено успорење или прекид протока података*	*Корисник може детектовати грешку коришћењем опција софтвера*
Велики утицај на проток података / Средње велика могућност детекције (СД5)	$\left(\begin{matrix}(0.5,0.6,0.7,0.8;\ 1,1)\\ ,(0.55,\ 0.6,0.7,0.75;\ 0.8,0.8)\end{matrix}\right)$	*Немогућност протока и обраде података у процесу*	*Детекција се може остварити скриптованом или захтеваном обрадом, корисник има информацију преко корисничког интерфејса*
Врло велики утицај на проток података / Велика могућност детекције (СД6)	$\left(\begin{matrix}(0.6,0.7,0.8,0.9;\ 1,1)\\ ,(0.65,\ 0.7,0.8,0.85;\ 0.8,0.8)\end{matrix}\right)$	*Немогућност протока података који су услов за друге протоке и обраде података*	*Детекција се обавља аутоматском тригерованом софтверском контролом*
Екстремно велики утицај на проток података / Апсолутна	$\left(\begin{matrix}(0.7,0.9,1,1;\ 1,1)\\ ,(0.75,\ 0.9,1,1;\ 0.8,0.8)\end{matrix}\right)$	*Прекид протока података који блокира све остале*	*Апсолутна ефикасност детекције, детекција је уграђена у*

ефикасност детекције (СД7)		*протоке података система*	*систем као део процеса*

Табела 7: Учесталост настајање грешака

Лингвистички искази	*IT2TrFNs*	*Опис за учесталост*
Веома ретко (О1)	$\left((0,0,0.15,0.25;1,1),(0,0,0.15,0.2;0.8,0.8)\right)$	*Грешка се дешава једном у току године*
Ретко (О2)	$\left((0.1,0.2,0.4,0.5;1,1),(0.15,0.2,0.4,0.45;0.8,0.8)\right)$	*Мање од пет пута годишње*
Периодично (О3)	$\left((0.3,0.4,0.6,0.7;1,1),(0.35,0.4,0.6,0.65;0.8,0.8)\right)$	*До једном у току дана*
Често (О4)	$\left((0.6,0.7,0.9,1;1,1),(0.65,0.7,0.9,0.95;0.8,0.8)\right)$	*Више пута дневно, до једном у току сата*
Веома често (О5)	$\left((0.75,0.85,1,1;1,1),(0.8,0.85,1,1;0.8,0.8)\right)$	*Грешка се дешава више пута у току сата*

3.5.4.4. Предложени Алгоритам

Ранг идентификованих грешака софтвера је одређен применом конвенционалне COPRAS методе [53]. На основу приоритета грешака доносиоци одлуке могу да одреде врсту и приоритет менаџмент иницијативе које треба да се предузму у циљу елиминисања идентификованих грешака софтвера. Предложена процедура за одређивање ранга грешака софтвера је надаље приказана.

Корак 1. Фази процена вредности РФ-а за за сваку идентификвану грешку, $\tilde{\tilde{v}}_{j\,k}$, $j = 1,..,J; k = 1,..,K$ је извршена од стране доносилаца одлука применом консензуса.

Корак 2. Отежана фази матрица одлучивања, $\left[\tilde{\tilde{d}}_{jk}\right]_{JxK}, j = 1,..,J; k = 1,\ldots,K$ је постављена. Вредности елемената ове матрице се одређују на основу правила фази алгебре [44]:

$\tilde{\tilde{d}}_{jk} = \tilde{\tilde{w}}_k \cdot \tilde{\tilde{v}}_{jk} = \Big(\big(a_k^U \cdot a_{jk}^U, b_k^U \cdot b_{jk}^U, c_k^U \cdot c_{jk}^U, d_k^U \cdot d_{jk}^U;\ \alpha_1, \alpha_2\big), \Big(\big(a_k^L \cdot a_{jk}^L, b_k^L \cdot b_{jk}^U, c_k^L \cdot c_{jk}^U, d_k^L \cdot d_{jk}^U;\ \beta_1, \beta_2\big)\Big)\Big)$ где је вектор тежина РФ-а означен као $\tilde{\tilde{w}}_k, k = 1,..,K$; поступак одређивања вектора тежина приказан је у претходној Секцији.

Корак 3. Репрезентативни скалари IT2TrFN $\tilde{\tilde{d}}_{jk}, d_{ik},\ j = 1,..,J;\ \ k - 1,..,K$ су добијени применом поступка дефазификације предложен у [52].

Корак 4. На нивоу сваке идентификоване грешке j, j=1,…,J одређују се агрегиране вредности бенефитних РФ-а, P_jи трошковних РФ-а, Q_j, аналогно процедури која је развијена у традиционалној COPRAS [53]:

$$P_j = \sum_{k=1}^{K'} d_{jk} \text{ и } Q_j = \sum_{k=K'+1}^{K} d_{jk},\ \ j = 1,..,J$$

Корак 5. Укупна агрегирана вредност која се придружује свакој идентификованој грешци j, j=1,…,J, се рачуна према изразу:

$$Z_j = P_j + \frac{Q_{\min} \cdot \sum_{k=K'+1}^{K} Q_j}{Q_j \cdot \sum_{k=K'+1}^{K} \frac{Q_{\min}}{Q_j}},\ j = 1,..,J;\ \ k = 1,\ldots,K$$

где $Q_{\text{мин}} = \min_{j=1,..,J} Q_j, j = 1,..,J$

Корак 6. Одредимо меру корисности за сваку грешку j, j=1,..,J тако да:

$$\zeta_j = \frac{Z_j}{Z_{\text{мах}}}, j = 1,..,J$$

где $Z_{\text{мах}} = \max_{j=1,..,J} Z_j, j = 1,..,J$

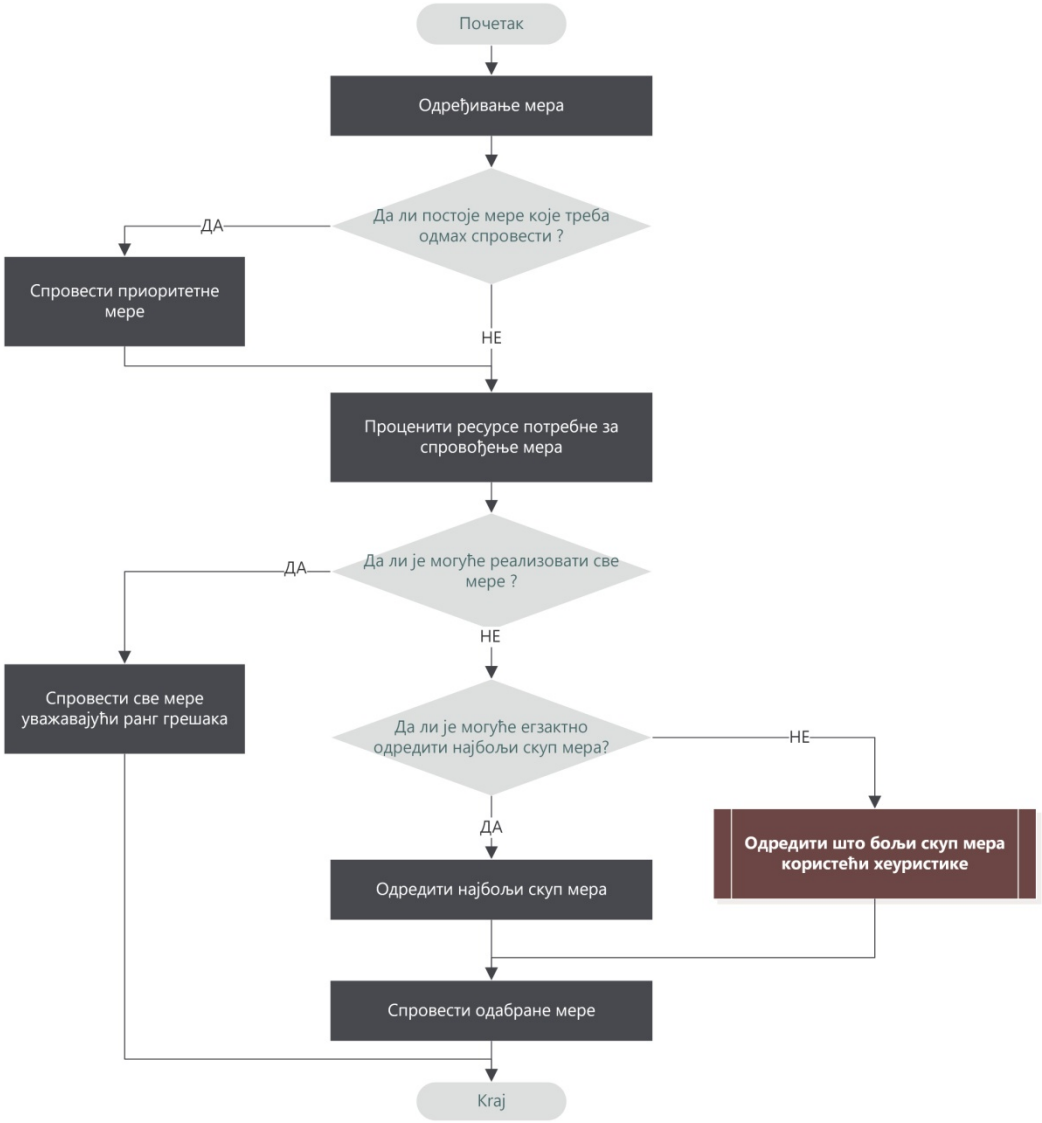
Корак 7. Ранг свих идентификованих грешака на нивоу сваког примитивног процеса се одређује респектујући израчунате вредности ζ_j. На првом месту у рангу, односно на последње месту у рангу, респективно налази се она грешка којој је придружена највећа, односно најмања вредност ζ_j, респективно.

Корак 8. Према добијеном рангу, доносиоци одлуке предузимају одговарајуће менаџмент мере у циљу смањења или елиминисања идентификованих грешака.

3.5.5. Идентификовање мера за отклањање грешака софтвера

Овај поступак представља креативну активност која се спроводи као наставак претходно спроведене модификоване FMEA анализе. Користи се познавање развијеног софтвера и информационог система. У тиму који спроводи овај поступак налазе се обавезно и чланови тима за развој софтвера.

Најпре се анализирају грешке и одређују мере за отклањање грешака, а потом се врши избор мера које ће бити спроведене. На слици 10 представљен је алгоритам по којем се спроводи одређивање и спровођење корективних мера.



Слика 10: Дијаграм тока одређивања и спровођења мера

При одређивању мера које могу да смање ниво ризика који настаје услед материјализације идентификованих грешака, узимају се у обзир идентификовани узроци грешака из FMEA анализе. У овом поступку анализом узрока грешака, на основу знања и искуства из претходно реализованих активности развоја посматраног софтвера, дефинишу се мере као могућа побољшања, модификације, дораде и исправке актуелне верзије софтверског решења. Ове мере ако се реализују треба да спрече нежељена стања и активности која представљају узроке идентификованих грешака. Мере које треба да остваре митигацију ризика, могу и треба да буду повезане са фазама софтверског процеса. Овако повезивање представља мапирање мера у односу на активности софтверског процеса и омогућава планирање и организацију нове итерације развоја софтвера.

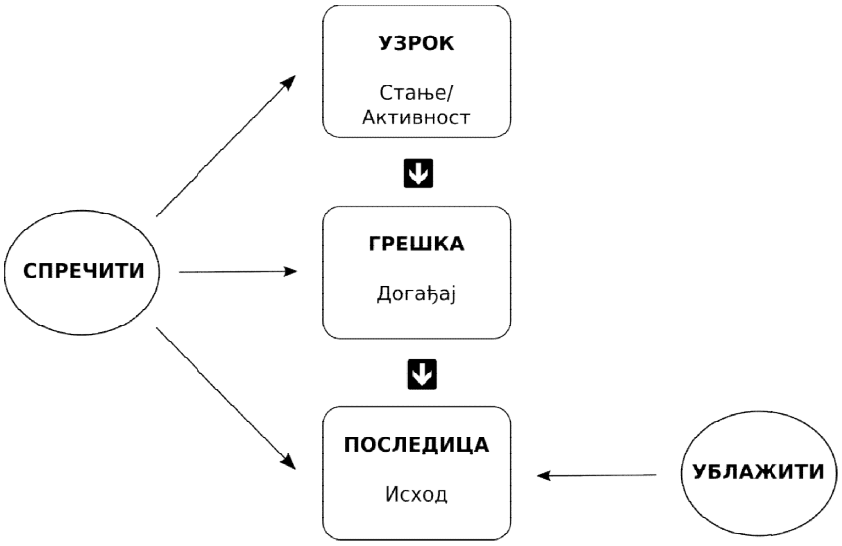
Како се у свакој итерацији развоја софтвера понављају фазе софтверског процеса, већ према изабраном моделу развоја, тако и мере долазе у разне фазе развоја. На развојном тиму је да процени где је неопходно обавити модификације решења. Ако претпоставимо да имамо фазе анализе, дизајна, имплементације и примене у софтверском процесу, и да се одвијају баш таквим редоследом, и ако знамо да постоји условљеност између активности у различитим фазама, треба тражити решење проблема у што каснијим фазама ако је то могуће. Модификације захтева вероватно ће значити прилагођавање дизајна. Другачији дизајн дела система, сигурно намеће поновно писање неког дела кода и слично. Једна мера тада води другој мери у каснијој фази. Примери различитих мера у разним фазама софтверског процеса приказани су у табели 8.

Табела 8: Примери мера за отклањање грешака у раду софтвера

Фаза софтверског процеса	**Активност**	**Узрок грешке**	**Предвиђена мера**	**Ко спроводи?**
Инжењеринг захтева	Анализа и валидација захтева	Различит начин прорачуна просечне оцене студента	Отклонити супротављене елементе захтева	Инжењер захтева, аналитичар
Анализа	Анализа	Коришћење	Ремоделовати	Пројектант

	система	неажурног податка	пословни процес	процеса
Дизајн	Дизајн модела података	Неусаглашеност кумулативног податка и појединачних ставки	Уклонити кумулативни ентитет, применити изведени поглед	Дизајнер модела података
Имплементација	Израда интерфејса	Фајл за размену није компатибилан	Користити старију верзију типа фајла	Апликативни програмер
Имплементација	Израда корисничког интерфејса	Улазни податак није исправан	Поставити контролу на форму за унос података	Фронтенд програмер
Имплементација	Кодирање	Модул за обраду оцене испаљује изузетак	Исправити део кода који је хардкодован и игнорише параметре	Апликативни програмер
Примена	Подешавање	Унос нетачних децималних бројева	Подесити параметре оперативног система	Администратор система
Примена	Обука корисника	Грешка при избору типа извештаја	Упутити кориснике у процедуру	Инжењер одржавања

Ако је потребно модификовати и пословне процедуре или начин рада корисника софтвера, неопходна је сарадња са менаџментом и стејкхолдерима. Као што је приказано на слици 11, осим мера за спречавања узрока, испољавања и последица грешака, могуће је одредити и мере које могу да ублаже последице грешака.



Слика 11: Начин деловања мера

За сваку меру треба такође одредити и потребне ресурсе (људски, временски, финансијски, технички и сл.) за спровођење мера. Одређивање потребних ресурса ослања се на раније реализоване активности и фазе софтверског процеса у којима се планира реализација одређених мера.

Ако није могуће реализовати све дефинисане мере тада је неопходно да се изврши избор предложених мера које ће се реализовати респектујући постојећа ограничења. Скуп свих претходно одређених потенцијалних мера, заједно са процењеним трошковима ресурса за примену и припадајућим ранговима грешака, представљају базу за одлучивање. Овај проблем може да се разматра као проблем математичког програмирања. Решење овог проблема може да се нађе применом различитих хеуристичких и метахеуристичких метода.

У случају посматрања једног ограничавајућег ресурса, најчешће је то у пракси временски ресурс, проблем се своди на:

Ако су дати:

рангови $r_0, r_1, \dots r_{n-1}$

трошкови $w_0, w_1, \dots w_{n-1}$

и капацитет W

Треба одредити н-торку

$$[x_0, x_1, \dots x_{n-1}] \in \{0,1\}^n$$

такву да

$$R = \sum_{i=0}^{n-1} r_i x_i \rightarrow max$$

$$\sum_{i=0}^{n-1} w_i x_i \leq W$$

За мале димензије проблема може се применити исцрпна претрага, или нека детерминистичка техника. Како простор решења садржи 2^n вектора, где је n број мера, већ за n=25 простор решења броји 33554432 могућа решења, а за н=50 треба претражити више од 10^{15} решења.

Ограничења у ресурсима, посебно временско ограничавање за примену мера, последица је итеративности и развоја у циклусима који требају бити добро одмерени како би се што пре дошло до позитивних ефеката, стога је битно оптимално одредити скуп мера.

3.5.6. Оцена ефеката предузетих мера

Описана модификована FMEA, односно анализа грешака софтвера информационог система, спроводи се након дефинисаног временског периода фазе одржавања посматраног софтвера. Уколико је раније спроведена FMEA анализа, најпре се реализује поређење ризика пре и након спроведених мера.

Такође, реализује се анализа ефеката протока података у процесима, пре и после већ реализованих мера. Користе се подаци из базе података о реализованим

обрадама података у посматраном контролном периоду после примењених корективних мера и увођења нове верзије софтвера. За сваки појединачни посматрани процес из CCA који је обухваћен у FMEA, анализира се разлика између вредности индикатора за период пре и после реализованих поступака. Очекује се да ће вредности дефинисаних индикатора за мерење ефеката протока података, бити побољшане у свим или већини посматраних процеса.

4. СОФТВЕР ЗА ПОДРШКУ ПРОЈЕКТОВАНОЈ МЕТОДОЛОГИЈИ

Да би се успешно практично применила предложена методологија, која повезује различите активности, прикупљање и обраду велике количине података, анализу ризика, интегрисаност у софтверски процес, коришћење фази логике, методе одлучивања и учешће потенцијално већег броја учесника, неопходно је користити софтверске алате за подршку. Коришћење таквих софтерских алата за подршку методологији која се доминантно бави анализом софтвера је пример CASE приступа. Да ли ће се користити готови софтвери, комерцијални или open-source, да ли ће се развијати сопствена решења или комбинација ових приступа, ствар је процене и одлуке оних који спроводе методологију. У овом истраживању планиран је у највећој мери сопствени развој нових и прилагођених софтверских алата.

Коришћењем различитих алата и средстава може да се повећа интеграција корективних и иновационих процеса. Добри алати могу да повећају ефикасност и висок степен аутоматизације предвиђених активности што доприноси повећању задовољства корисника. Често је потребно као и у овој методологији коришћење резултата из једне методе као улазних података за другу методу што може бити велика предност коришћења софтверских алата. И поред свега наведеног не треба заборавити да ипак све зависи од људског фактора и правилног коришћења софтвера. Корисници морају бити стручни и обучени за употребу рачунара и за коришћење софтверског решења као и за примену метода и поступака у обради података и информација.

Узимајући у обзир чињеницу да је FMEA веома захтевна метода за спровођење, многи аутори предлажу коришћење софтверске подршке. Такав софтвер би требао смањити утрошено време, олакшати унос података, обезбедити проверу тачности, поједноставити поступке, интегрисати парцијална решења.

Постојање оправдане потребе за таквим софтверским решењима, а посебно заснованим на знању, потврдили су у својој студији Johnson i Kan [78]. У

претходном периоду било је различитих софтверских приступа. У [79] су за потребе рачунарске симулације процеса, повезали FMEA процес и различите технике вештачке интелигенције. У [80] су радили на аутоматизацији FMEA и функционалном моделирању. Развили су софтверски прототип FMEA система. Приказали су функционално моделирање система засновано на резултатима добијених од темељног структурног симулатора [80]. У [81] су понудили WIFA пројекат који се темељи на бази знања [81]. Huang i Mak су предложили занимљив софтверски FMEA приступ заснован на прикупљању података на интернету. Овакав приступ заснован на Web-у осим побољшања FMEA омогућио би и укључивања пословних партнера у реализацију методе [82]. Систем за контролу квалитета динамичких ланаца снабдевања уз коришћење рачунарске обраде података и графичког корисничког интерфејса је предложен у [83].

4.1. Структура софтверског решења

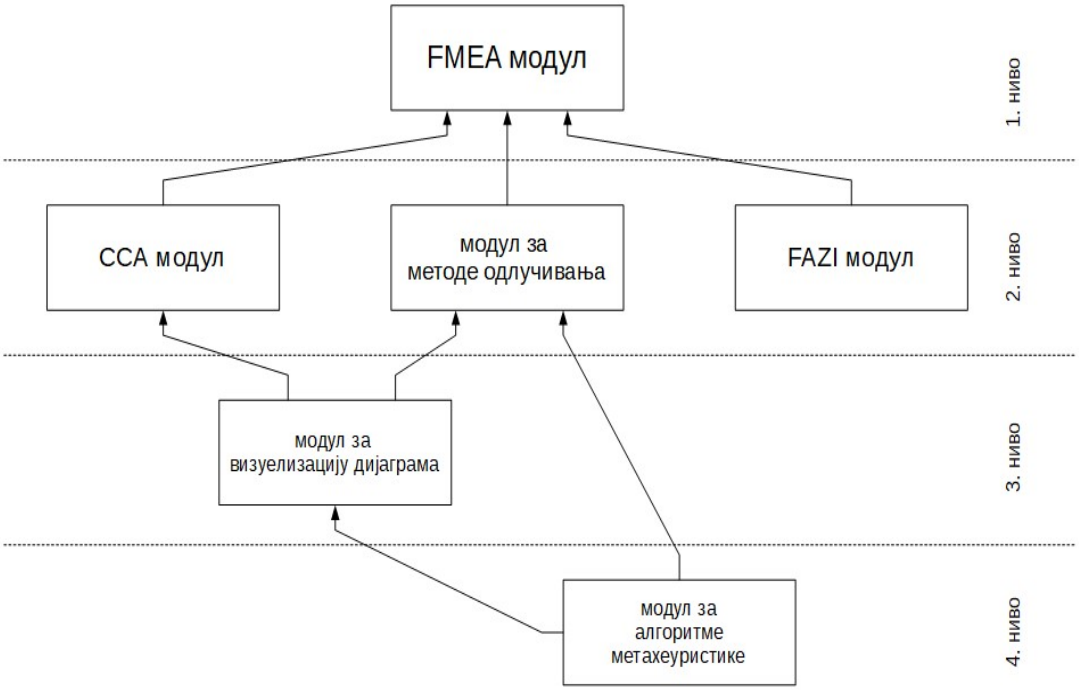
Током експерименталне примене предложене методологије, развијена су одређена софтверска решења без којих би било веома тешко спровести потребне активности, или би то било реализовано неефикасно и уз велику потрошњу ресурса. Да ви се обезбедило коришћење софтвера за подршку методологији, примењен је модуларни приступ развоју. Развој почетних верзија одређених софтверских модула омогућио је брзу примену ових алата у неким деловима реализације предложене методологије. Развијање засебних модула омогућава флексибилно комбиновање софтверских блокова за разне примене и сценарије. У току овог истраживања, захваљујући модуларном концетпу и комуникацији модула према строго дефинисаним интерфејсима, било је могуће користити компоненте различитог нивоа завршености и константно укључивати нове верзије појединачних модула у заједнични систем. Овакав начин развоја наставиће се и након овог истраживања до комплетирања делимично завршених делова система.

Структура софтверског система за подршку предложеној методологије састоји се од следећих софтверских модула:

- Модул за универзалну FMEA
- Модул за CCA
- Модул за методе одлучивања
- Модул за ФАЗИ моделе
- Модул за алгоритме метахеуристике
- Модул за визуелизацију дијаграма

Међусобни однос софтверских модула у конкретној примени у оквиру ове дисертације представљен је на слици 12. Структура у којој ће бити позициониране софтверске компоненте зависи од потреба у конкретном пројекту. Распоред елемената на приказаном дијаграму показује однос софтверских модула и њихов значај за примену у овом истраживању. На врху се налази као кровни, модул за универзалну FMEA. Овај модул који је позициониран у први ниво на дијаграму, користи модуле на другом нивоу: модул за CCA, модул за методе одлучивања и

модул за фази моделе. За мање значајне у овом случају, али како се показало корисне, сматрају се компоненте које обезбеђују функционалности визуелизације и имплементацију метахеуристичких алгоритама. Овакво софтверско решење је веома флексибилно и омогућава додавање нових компоненти решења, као и једноставну измену раније уграђених.



Слика 12: Структура софтверског решења - модули решења

Пројектовање софтвера подразумева дефинисање скупа компоненти које ће бити креиране и од којих ће се састојати будући систем. Осим скупа компоненти потребно је и одредити везе између њих. У овом пројекту је примењен метод модуларизације на функционалној основи. Издвојене су компоненте тако да свака групише своје сродне функционалности. Најпре су дефинисане основне функционалности на нивоу компоненте, а потом су разрађене детаљно. Parnas је заслужан за увођење модуларизације у развој софтвера, указао је да је

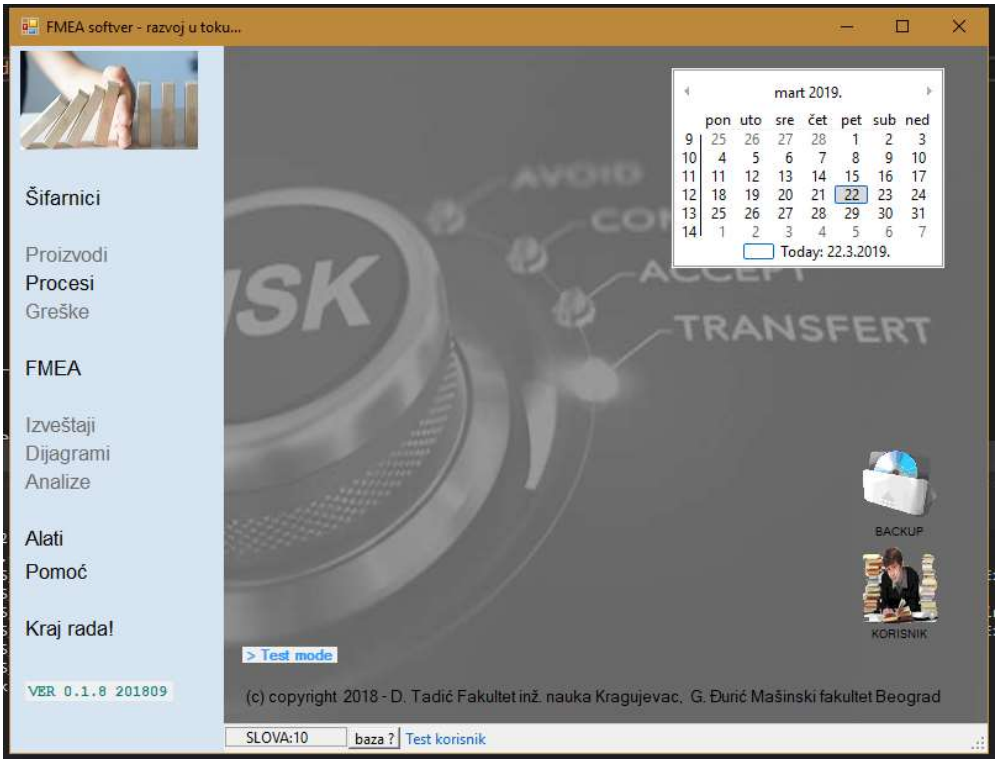
модуларизација начин за повећање разумљивости система као и флексибилности која омогућава једноставније проширивање и одржавање [84].

Декомпоновање система на модуле има за циљ стварање компоненти које ће се независно развијати, које се могу комбиновати и вишеструко користити, које се могу мењати без утицаја на друге компоненте. Коришћењем функционалне декомпозиције формирана је приказана хијерархија модула где се сваки следећи ниво има већи приступ информацијама од претходног.

4.2. Елементи софтверског решења

4.2.1. Универзална FMEA апликација

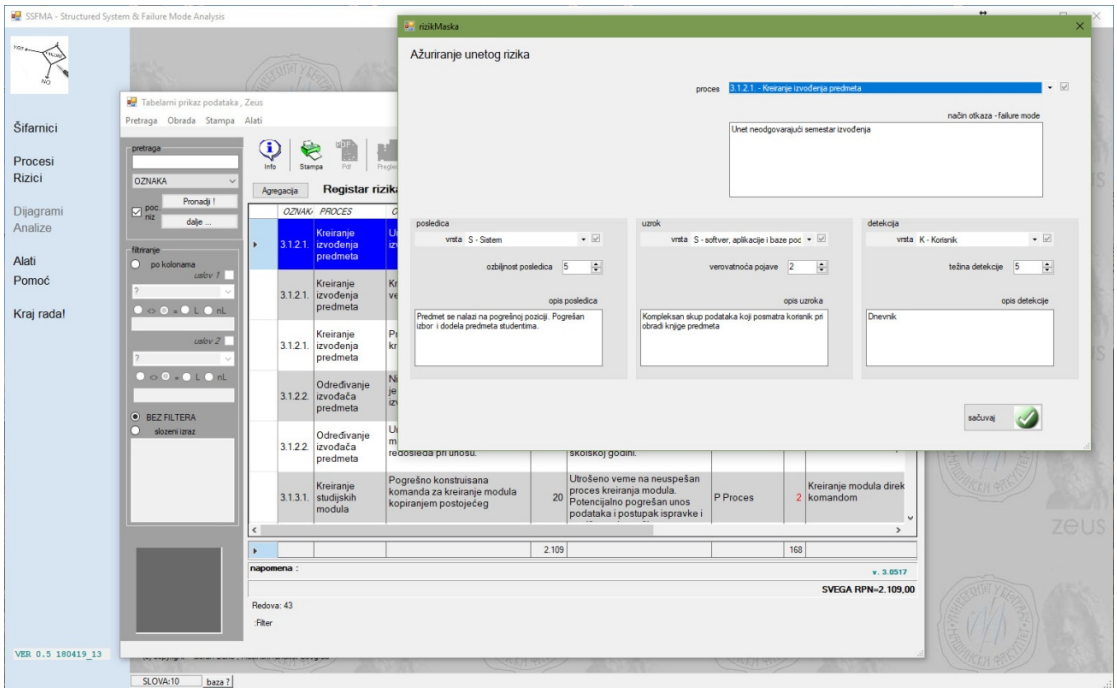
Најважнији део овог софтверског решења је модул за FMEA. Представља кровну компоненту која повезује све остале тренутно дефинисане модуле. Реализована је као софтверска апликација. Пошто повезује скоро све елементе методологије, може се рећи да је у питању кључни CASE алат у овом истраживању. Мада је основни мотив за развој ове апликације потреба за софтверском подршком при спровођењу предложене методологије у овом раду, као један од основних захтева постављен је захтев високог степена универзалности решења. Основни екран са ставкама првог нивоа менија апликације приказан је на слици 13.



Слика 13: Основни екран и мени апликације за FMEA

Решење је имплементирано у облику десктоп апликације за оперативни систем Windows. Апликација је класично клијент-сервер решење са вишеслојним дизајном. Коришћен је програмски језик C# за програмирање пословне логике и елемената корисничког интерфејса. У питању је објектно-оријентисан програмски језик широке намене, погодан за примену пројектних шаблона. Коришћење пројектних шаблона приликом пројектовања и имплементације овог софтвера је одлука која је помогла не само да се користе добре програмерске праксе већ и као средство ефикасне комуникација сарадника на високом нивоу апстракције, без губитка времена на мање битним детаљима. База података је реализована као SQL база података Oracle 11g express. У питању је слободна за коришћење база података са малим отиском и са солидним сетом администраторских опција.

Апликација излаже богат, доследан, конзистентан графички кориснички интерфејс. Осим уобичајене парадигме и манипулације графичким елементима имплементиран је и текстуални интерфејс погодан за брз унос података. На слици број 14, приказан је један типичан радни екран апликације.



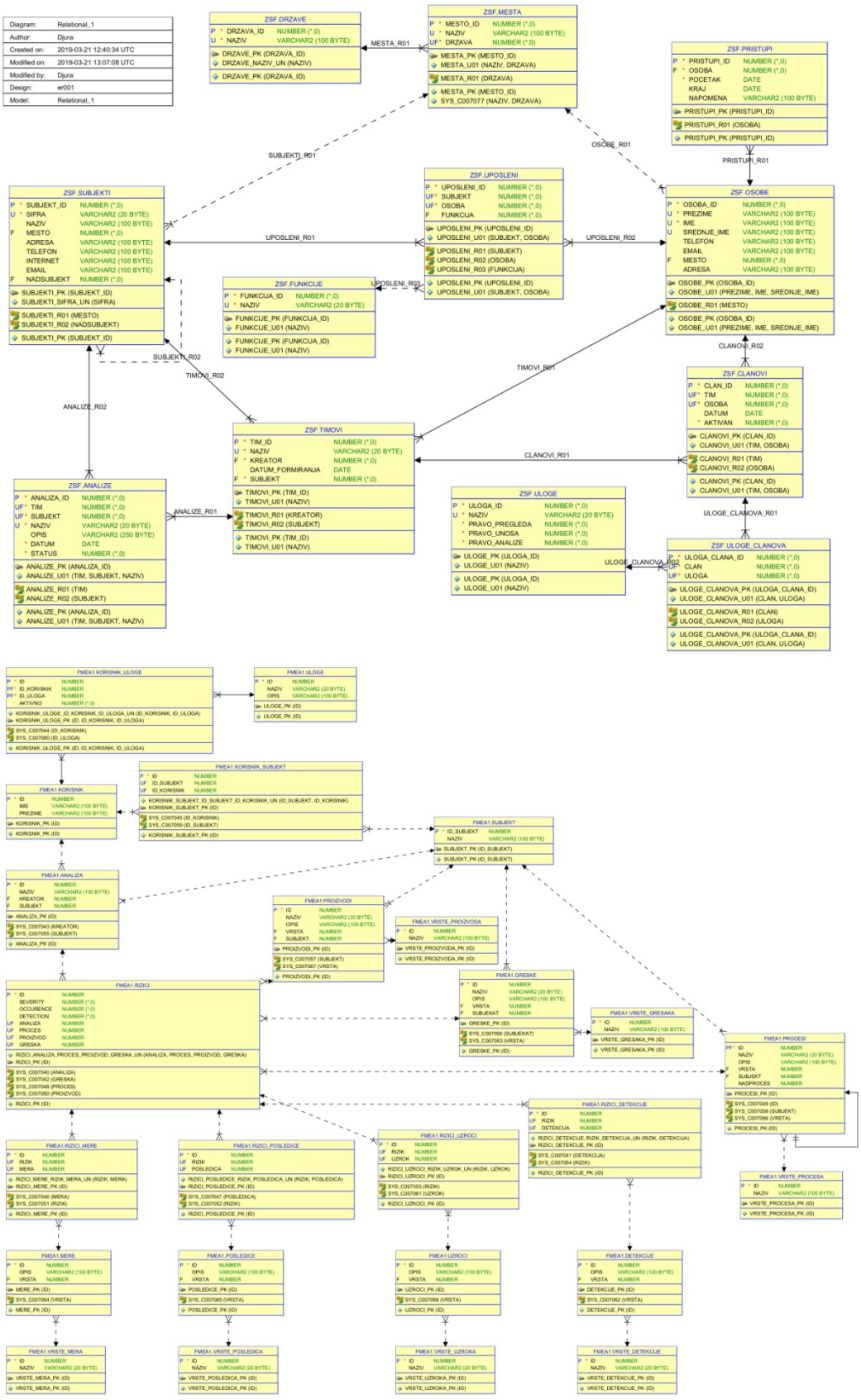
Слика 14: Радни екран апликације за FMEA

Основне функционалности тренутне верзије апликације су: управљање шифарницима и класификацијама, евиденција производа и/или материјалних ствари, евиденција процеса, евиденција врста грешака/отказа/одступања, управљање и унос података FMEA, генерисање извештаја, генерисање графичких извештаја, креирање анализа, администтзрација система.

За спровођење FMEA потребни су људски ресурси и треба омогућити тимски рад. Апликација омогућава неометан вишекориснички рад над заједничким сетом података.

Најважније могућности апликације концентрисане су на основне операције са креирањем, уносом ставки, манипулацијом и прегледом података FMEA. Као део ширег софтверског решења ова апликација осим језгра својих функција садржи као веома важан део методе комуникације са другим софтверима. Овај интерфејс служи за комуникацију и размену података са модулима софтверског решења и са екстерним софтверима. Начин размене података је реализован кроз импорт и експорт података у фајловима, што је примењиво и на интерне модуле решења и на спољне софтвере, и преко заједничке базе података што је резервисано само за модуле решења. Користе се стандардни формати фајлова: xml формат за пренос података ка системима релационих база, xls формат за потребе преноса ка апликацијама за презентацију и канцеларисјку обраду, csv формат за потенцијалну размену са старим системима. У постојећој конфигурацији система за потребе спроведеног истраживања на факултету, као улазни споља формирани скуп података за ову апликацију послужио је скуп података са хијерархијом процеса студентске службе формиран у другом софтверу - модулу за CCA који је део софтверског решења.

Респектујући очекивани унос великих количина података, као и високо позициониран захтев да решење буде универзално примењиво за спровођење анализе ризика FMEA у различитим делатностима и системима, посебна пажња је посвећена пројектовању адекватног модела података као основе овог решења. Актуелна верзија модела података која је коришћена у току спроведеног истраживања перформанси процеса информационог система Студентске службе, приказан је на слици 15 у облику ER дијаграма.



Слика 15: Модел података Multitenant-FMEA

Модел осим основних ентитета: анализа, предмет, процес, човек, тим, субјект и грешка, обезбеђује услове за произвољан број и структуру атрибута основног ентитета. На тај начин створени су услови за висок степен универзалности решења које користи складиште података креирано према моделу. Модел омогућава унос података о различитим анализама, различитог типа, за различите организације, спроведених од различитих тимова, све међусобно повезано и поређано по потреби. Приказана верзија модела је компромис између жељене универзалности и ширине модела и потребе да се физичка имплементација модела на конкретан дизајн реализује са коришћењем свих предности релационог модела података и гаранцијама интегритета података и референцијалног интегритета. Тренутна верзија модела је омогућила креирање multi-tenant устројене базе. Multi-tenant приступ даје могућност симултаног рада више различитих корисника на истој платформи. Предности оваквог приступа су скалабилност система, централизација управљања, стандардизација процеса и оптимално коришћење ресурса. Уобичајена пракса је пресликати више појединачних логичких шема у апликацији у једну физичку шему више субјеката у бази података, оваква пресликавања је изазовно направити због могућих накнадних проширивања основних шема [85].

Развијена апликација као и већина апликација које користе базе података има више логичких слојева: слој података где се чувају потребни подаци, слој корисничког интерфејса који омогућава комуникацију корисника и апликације и слој пословне логике који управља процесима и повезује остале слојеве. Вишеслојни приступ развоју апликација има бројне предности: умањује сложеност система, припрема систем за евентуалне модификације, повећава приступачност у односу на клијент/сервер апликације и пружа велику сигурност због индиректног приступа подацима [86].

Осим имплементираног модела података и модуларног концепта структуре решења, омогућена је конфигурабилност рада система заснована на сетовима података - параметрима. Параметризација апликације омогућава фино подешавање од стране корисника. FMEA метода се базира на искуству, знању и креативности људи који је спроводе. Прилагођањем и адаптирањем апликације

кроз дефинисање параметара, креирање наменских шифарника, класификација, лингвистичких скала и разних других елемената потребних за FMEA анализу у различитим делатностима, софтверско решење се може прилагодити било којем случају коришћења.

Конфигурисање апликације за примену у конкретном истраживању, подразумевало је осим успостављања интерфејса са модулом за ССА и дефинисање одређених класификација и лингвистичких исказа према предложеној методологији и према искуствима из развоја посматраног софтвера информационог система Студентске службе и сличних пројеката. Део података из дефинисаних иницијалних шифарника је приказан у табели 9.

Табела 9: Шифарници иницијалног IT сета за FMEA апликацију

класификација	**ознака**	**опис**
врста последице	P	унутар процеса
	S	системски
врста узрока	O	организациони фактори, процедуре
	K	људски фактор, корисници система
	S	софтвер, апликације и базе података
	H	хардвер, мрежа, IT инфраструктура
врста мере за детекцију	K	корисници
	S	софтвер
фаза софт.процеса	Z	захтеви
	A	анализа
	D	дизајн
	I	имплементација
	O	одржавање

Развој ове апликације и софтверског решења шире посматрамо, као и сваки развој софтвера, не завршава се завршетком овог истраживања. Тачнија је оцена да је ово само прва почетна фаза развоја. План даљег развоја је усмерен најпре на

комплетирање почетних верзија незавршених модула решења. На функционалном нивоу, планира се омогућавање израде симулација. Свака симулација садржи суштински три елемента. То су стварни систем - домен, модел који је апстракција домена и активни елемент - симулатор, који је у нашем случају софтвер. Део елемената тренутна верзија софтвера већ садржи. Планира се израда модула који ће бити доступан преко интернета. То ће бити део решења реализован као скуп web сервиса и постављен у облаку. Рачунарство у облаку (енглески Cloud computing) представља пружање услуге коришћења рачунарских ресурса. Рачунарство у облаку, има потенцијал за трансформацију великог дела ИТ индустрије чинећи софтвер још атрактивнијим представљајући га као услугу [87]. Примена ове технологије у верзији хибридног облака, где ће систем омогућити приступ многим организацијама, довешће до сарадње тимова, и дељења информација. Уносом података из разних делатности и организација, може доћи до формирања јединствене базе знања примењиве за различите анализе и ситуације.

4.2.2. Модули софтверског решења

4.2.2.1. Модул CCA

Први софтверски модул који је развијан у овом истраживању био је модул за подршку спровођења структурне системске анализе. Потреба за евидентирањем идентификованих процеса и њихово повезивање у хијерархијску структуру решени су као прве функционалности овог софтверског решења. Реализован је као софтверска апликација. Избор технологија за имплементацију овог дела система је идентичан као за FMEA апликацију. Омогућен је унос података кроз графички кориснички интерфејс али и преко xls фајлова ако корисник жели да користи неки кориснички интерфејс са којим је фамилијаран.

Основни функционални захтеви постављени пред овај софтвер одговарају опису суштинског резултата сваке комплетно спроведене ССА, а то су хијерархијски организован скуп дијаграма тока података, речник података свих токова и складишта података, модел података и спецификација логике свих примитивних процеса. У реализованој верзији апликације, највећа пажња посвећена је развијању технике уноса података који дефинишу дијаграм тока података (ДТП), који је основна парадигма методе. Као што је показано у методологији, могуће је сваки ДТП описати једноставним формалним записом. У оквиру ове апликације обезбеђен је интерфејс за брз и прецизан унос података у форми таквог записа. Програмска аутоматска провера исправности ДТП-а и провера хијерархије процеса и повезаности и логичне везе елемената из дијаграма различитог нивоа су се показале као корисне, доприносећи ефикасности активности које захтевају креативни напор од чланова тима за реализацију ССА. Потреба да се подаци који описују дијаграм меморишу решен је овом апликацијом, али потреба да се ти подаци прикажу графички, што је важан аспект ССА методе, решен је помоћу функционалности другог модула из овог софтверског решења - модула за визуелизацију дијаграма.

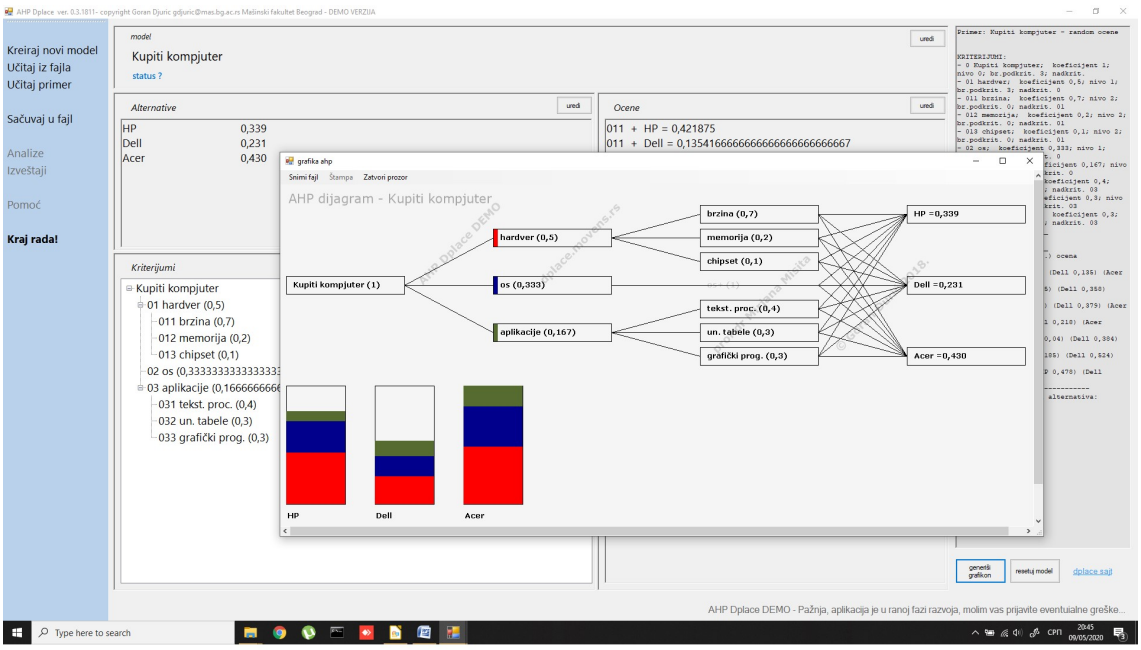
Списак процеса креиран за потребе истраживања у овој апликација као излаз из ње, послужио је као улаз података за други модул решења, за FMEA модул. У следећим итерацијама развоја овог софтвера, планира се развој функционалности размене модела података са трећим софтверима који служе за администрацију и програмирање релационих база података. План је да се оствари комуникација у оба смера, чиме би се убрзао процес моделовања. Такође, у случају спровођења ССА са постојећим активним решењима, ова размена модела, послужила би за снимање постојећег стања и реверзибилни инжењеринг.

4.2.2.2. Модул за методе одлучивања

Идеја у основи овог софтверског модула је издвајање и груписање функционалности оцењивања, рангирања и избора између могућих алтернатива.

Посебан нагласак стављен је на имплементацију различитих метода вишекритеријумског одлучивања. Вероватно је сваки проблем данашњице у основи вишекритеријумски. Решење готово сваког реалног проблема зависи од комбинације више критеријума оптималности. Овакви проблеми могу да се представе као задаци ВКО. У докторском раду имплементиране су методе AHP и COPRAS. Функционалности формирања алтернатива и одређивање критеријума оптималности би према почетним захтевима требало да се нађу изван овог модула.

Модул је реализован као библиотека класа DLL за .NET радни оквир. AHP метода је имплементирана као засебна компонента овог модула, као класа у библиотеци класа. Иницијално за потребе тестирања имплементиране методе, а касније и за коришћење у процесу наставе на факултету, развијена је засебна демонстрациона десктоп апликација као spin-off развоја модула. Пример радног екрана ове апликације која у основи представља танак слој корисничког интерфејса према методама класе из развијене библиотеке класа, приказан је на слици 16.



Слика 16: Радни екран демо AHP апликације

4.2.2.3. ФАЗИ модул

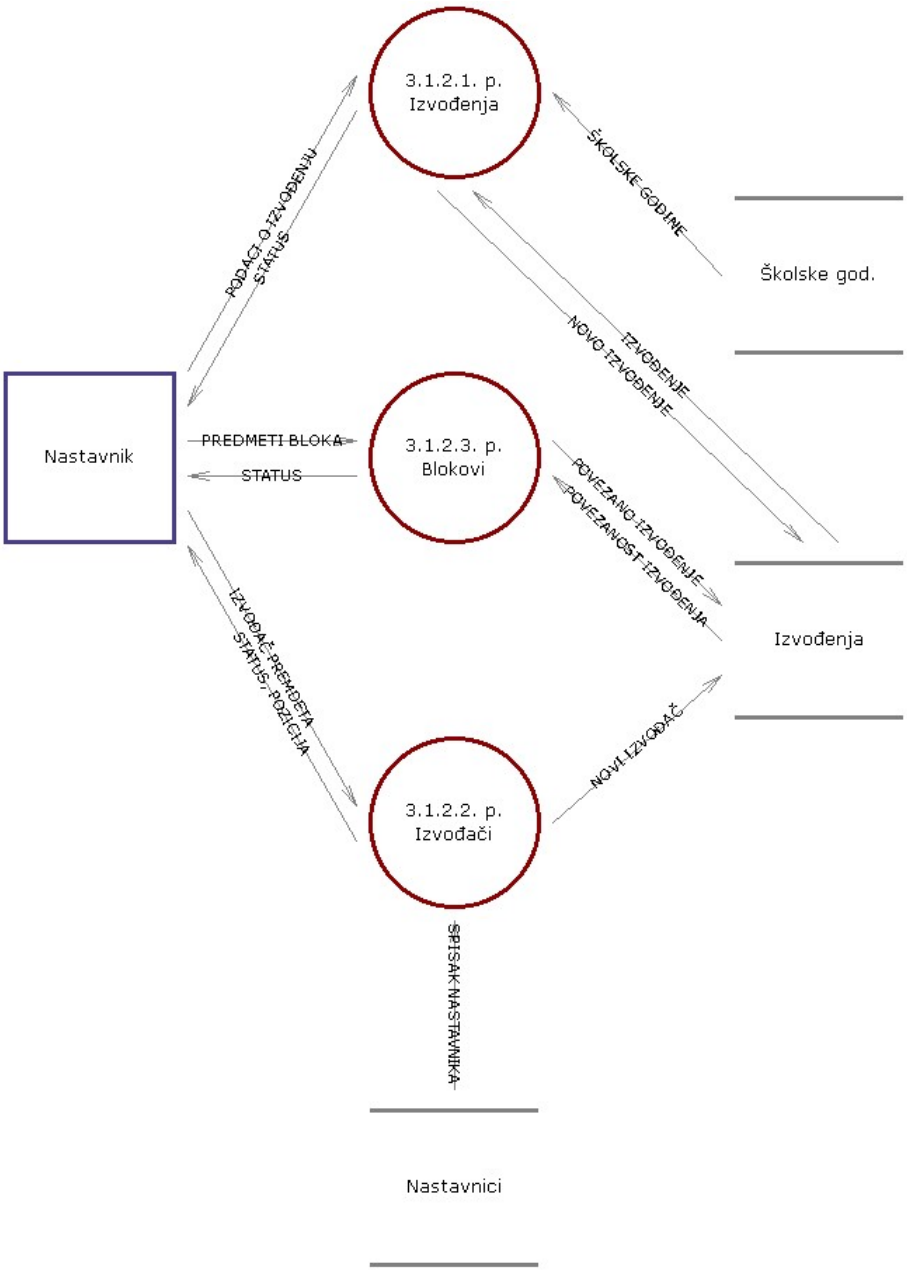
Овај модул софтверског решења обједињује функционалности које се односе на фази логику. Значај фази логике и велики потенцијал леже у чињеници да је људско размишљање по природи апроксимативно. Операције на фази бројевима, коришћење различитих облика функције припадности, методе дефазификације, методе поређења фази бројева и друге методе, саставни су део захтеваних функционалности овог модула. Модул се реализује као библиотека класа. У структури укупног решења овај модул излаже одговарајуће методе својих класа другим деловима софтверског система. Интерно обављање потребних, често рачунски захтевних операција са фази бројевима, посебно са тип-2 фази бројевима, биће једноставније, поузданије и робустније него коришћење екстерних софтверских алата.

4.2.2.4. Модул за визуелизацију дијаграма

Модул за визуелизацију дијаграма је издвојен из модула за подршку ССА чији је део иницијално био. Како је дијаграм тока података кључна парадигма ССА, потреба да се дијаграм брзо прикаже графички кориснику, осим ефикасног уноса записа у базу података, довела је до захтева за функционалношћу визуелизације. Пример генерисаног дијаграма приказан је на слици 17.

Корисник ни у једном тренутку не црта дијаграм, то треба да буде функција софтвера. Модул је реализован као библиотека класа. Актуелна верзија садржи методе за генерисање хијерархијског дијаграма за потребе АНР методе, и две врсте дијаграма тока података за потребе ССА. Излаз из ових метода може да буде PDF документ или PNG датотека. У току овог истраживања брза аутоматска визуелизацију дијаграма показала се као ефикасно средство провере унетих вредности, грешке при уносу су биле лако уочљиве на графичкој презентацији дијаграма. Последња имплементирана метода визуелизације дијаграма тока

података користи имплементацију генетског програмирања из модула за алгоритме метахеуристике.



Слика 17: Аутоматски генерисан дијаграм, метода 1

4.2.2.5. Модул за алгоритме метахеуристике

Овај модул групише функционалности које имају везе са применама алгоритама метахеуристике. Метахеуристика је хеуристика опште намене, то је у основи стратегија која управља поступком претраге. Модул је реализован као библиотека класа и тренутно садржи имплементацију генетског алгоритма и генетског програмирања који су део области еволутивног програмирања и засновани су на принципима природне еволуције и наслеђивања. Користе се доста успешно за решавање разних оптимизационих проблема. Специфичност генетског програмирања у односу на генетски алгоритам је што претражује простор решења у облику рачунарских програма - алгоритама за решење постављеног проблема. У оквиру предложене методологије, предвиђена је примена метахеуристичких алгоритама у ситуацијама када је немогуће другим методама одредити оптимално решење за решавање проблема избора оптималног подскупа мера за отклањање грешака софтвера. За примену генетског алгоритма за решавање овог комбинаторног проблема потребно је да развојни тим процени потенцијални ефекат на проток података система свих понуђених мера као и да процени трошкове реализације мера у смислу потребних ресурса.

5. ЕКСПЕРИМЕНТАЛНО ИСТРАЖИВАЊЕ

Експериментална провера предложене методологије реализована је у оквиру информационог система Службе за студентске послове Машинског факултета универзитета у Београду. Развој софтвера за подршку процесима студирања који је развијан на Машинском факултету у периоду од 2014. до 2019. године, изабран је за проверу методологије зато што је било могуће тесно повезати активности развоја и одржавања софтвера. Аутор овог докторског рада је истовремено и аутор наведеног софтверског решења.

Прикупљање, обрада, архивирање и анализа података су уобичајени задаци информационог система Службе за студентске послове (у даљем тексту студентска служба) факултета. ИС студентске службе је систем који служи за подршку процесу студирања на факултету. Садржи велики број процеса обраде података и сложене везе различитих елемената система. Као и свака друга организација и факултет има сопствене пословне процесе који обједињује у себи задатке и циљеве који треба да се остваре са једне стране и активности и ресурсе са друге стране који су потребни да се остваре ти задаци. Ови пословни процеси су неминовно повезани и у односу међузависности. Основни циљ развоја информационог система студентске службе је да се подрже суштински процеси који се одвијају на факултету и да се коришћењем информационих технологија и софтверским решењима олакшају и побољшају активности у вези наставе и студирања на факултету.

Овакав ИС, са процесима који сви обавезно обухватају неку обраду података и зависе од колекција података, може се звати и датацентричним или трансакционим информационим системом. Као такав био је погодан за примену предложене методологије.

5.1. Опис организације

Истраживање са применом предложене методологије у овој дисертацији, спроведено је у вези са развојем и одржавањем софтвера Студентске службе на Машинском факултету Универзитета у Београду.

Машински факултет Универзитета у Београду, као део групације техничко-технолошких наука, је најстарија је и највећа високошколска и научна установа у Србији у области машинства. На факултету је у току школске 2019/2020. године активно студирало преко 3000 студената. Факултет је у тренутку писања овог рада имао око 400 запослених, а од тога око 200 наставног кадра. Машински Факултет је високошколска установа која обавља образовну, научну и истраживачку делатност.

На Факултету се изводе академске студије на основу одобрених, односно акредитованих студијских програма за стицање високог образовања:

- Основне академске студије - Машинско инжењерство (ОАС), које трају три године, и чијим завршетком се стиче најмање 180 ЕСПБ бодова;
- Основне академске студије - Информационе технологије у машинству (ИТМ), које трају три године, и чијим завршетком се стиче најмање 180 ЕСПБ бодова;
- Мастер академске студије (МАС) - Машинско инжењерство, које трају две године, и чијим завршетком се стиче најмање 120 ЕСПБ бодова;
- Докторске студије (ДС) - Машинско инжењерство, које трају три године, и чијим завршетком се стиче најмање 180 ЕСПБ бодова.

Служба за студентске послове Машинског факултета Универзитета у Београду је посебна организациона јединица задужена за прикупљање, ажурирање, чување и обраду свих релевантних података о студентима и процесу студирања. Подаци, као и начин обраде података, су дефинисани законским и подзаконским актима.

5.2. Опис доменског софтвера

ИС Студентске службе служи за подршку активностима везаним за процес студирања на факултету. Сврха пројекта Зеус био је развој комплетног софтверског решења за управљање процесом студирања на Машинском факултету, са циљем да обезбеди испуњавање свих постављених функционалних и нефункционалних захтева, уз ефикасност рада свих укључених актера и висок квалитет излазних резултата.

Циљеви који су постављени пред ауторе софтверског решења су били:

- Решавање основних и критичних захтева;

- Решавање постављених функционалних захтева;

- Обезбеђивање извештаја и података дефинисаних законом и захтевима Универзитета као надређене организације;

- Ефикасност рада свих укључених актера;

- Максимална доступност информација свим стејкхолдерима;

- Могућност повезивања са другим информационим системима, посебно са Информационим системом универзитета;

- Висок квалитет излазних резултата;

- Аутоматизација поступака и активности;

- Приступ систему кроз разноврсне и савремене комуникационе канале;

- Механизми за квалитетно управљање системом;

- Елиминисање лоших процедура и решења;

- Решење засновано на дефинисаним захтевима;

- Потпуна контрола система и ефикасно одржавање;

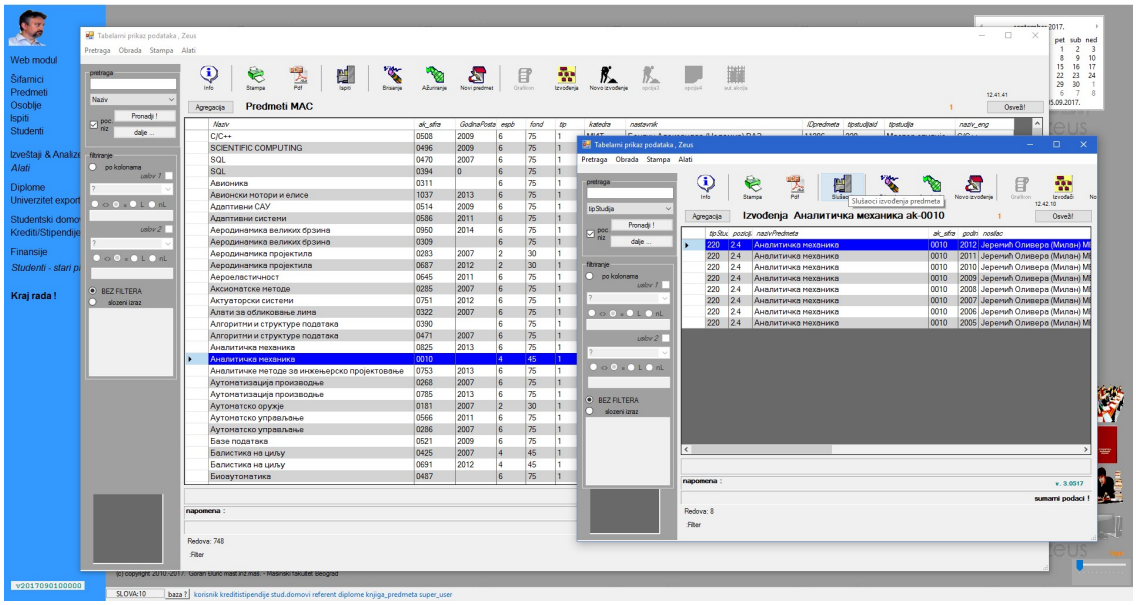
- Изворни код решења и документација на располагању факултету ;

- Постављање основе за даље интегрисање информационог система факултета;

- Ефикаснији рад служби и факултета у целини;

- Развој новог решења уз неометан рад постојећег система;

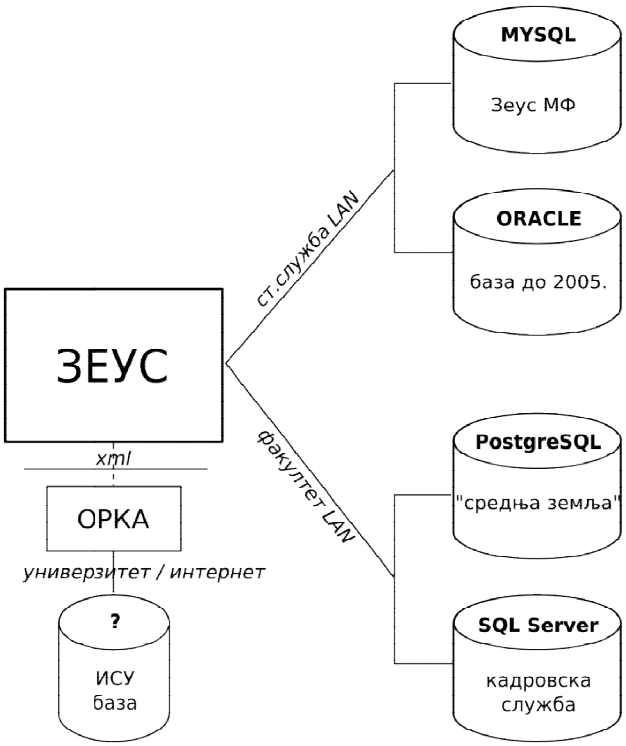
У структури решења, основни део решења реализован је као клијент-сервер систем који ради у локалној мрежи студентске службе. Најважнији део клијентског слоја је развијена desktop апликација. Ова апликација је опремљена са богатим корисничким интерфејсом као што се може видети на слици 18 где је приказан радни екран апликације са табеларним приказом података.



Слика 18: Зеус апликација

Основна база података система је реализована помоћу MySQL сервера базе података. Историјски подаци из претходних система смештени су у оквиру Oracle 9i базе података. Web компонента информационог система реализована је као засебна порталска интернет апликација - студентски сервис. У основи овог сегмента је PostgreSql сервер базе података и функционише већим делом као

реплицирани подскуп основне базе података на бази дневне ажурности. Софтвер имплементира управљање подацима у складу са релационим моделом података. Предвиђено је коришћење механизама тригера и трансакција. Захваљујући примењеном пројектном шаблону мост, омогућено је коришћење различитих система за управљање базама података као што је приказано на слици 19.



Слика 19: Коришћење различитих база података из Зеус апликације

Шаблони пројектовања су стандардан начин поновног коришћења кода између пројеката и између програмера. Користе се за систематизовање знања и доказаних добрих решења у разним инжењерским дисциплинама. Програмерски шаблон Мост је структурални шаблон. Структурални шаблони говоре о томе како формирати одређену погодну структуру састављену од класа и објеката. Шаблони заснован на класама најчешће користе наслеђивање да би се обезбедило жељено понашање система, а шаблони засновани на објектима говоре како формирати композиције објекта. Шаблон Мост (Bridge) раздваја апстракцију од реализације,

односно интерфејс од имплементације, тако да се могу мењати засебно. Овај моћан шаблон користи композицију објеката радије него наслеђивање класа, омогућава нам да логички у структури решења идемо у ширину а не само у висину.

Информациони систем универзитета садржи обједињене податке из евиденција свих припадајућих факултета. Ови подаци се воде се као јединствена база података. За потребе вођења евиденција, издавања јавних исправа, статистика и анализа у информационом систему универзитета посебно се прикупљају подаци о студијским програмима, кандидатима за упис, уписаним студентима, наставном особљу и ненаставном особљу. Машински факултет из свог информационог система експортује и формира скуп података који доставља информационом систему универзитета према детаљно дефинисаној спецификацији. Подаци се шаљу најмање једном дневно. На слици 20 представљен је дијаграм размене података делова ИС-а студентске службе.

Апликативни слој основног дела решења реализован је кроз следеће компоненте:

- Апликација Зеус, основна апликација за рад у студ.служби;

- Апликација МФ-ИСУ за размену података са РЦУБ-ом;

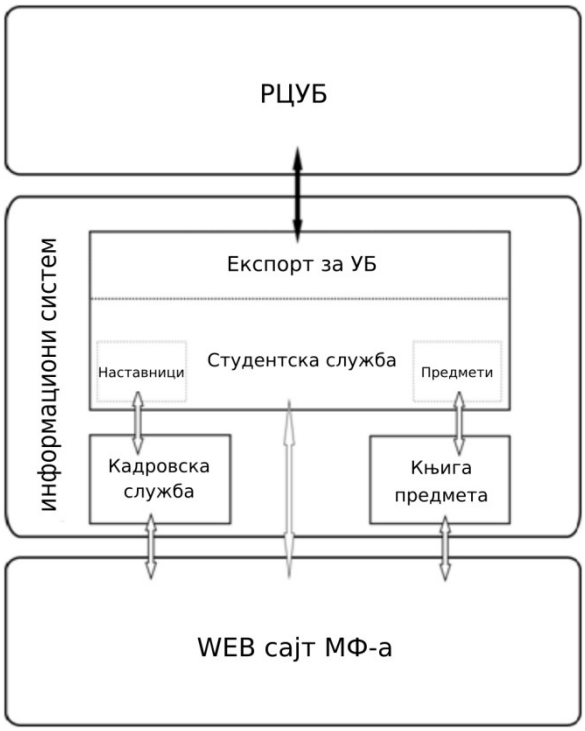
- Апликација за синхронизацију са web сајтом и сервисима ;

- Апликација Записник за унос оцена са испита;

- Апликација за штампу додатка за диплому;

- Библиотека заједничких класа интерфејса

- Библиотека Зеус-МФ класа решења



Слика 20: Размена података ИС-а студентске службе

Нова апликација у својој основној верзији уведена је у продукцију у јануару 2014. године. У склопу нове апликације решени су најважнији и критични функционални захтеви, као и нови захтеви пристигли у току ове фазе. Развијен је и имплементиран подсистем за Студентски сервис и генерално за потребе Web сајта факултета. Систем је функционисао са дневном ажурношћу. Активности у периоду од 2014. до 2016. године се могу грубо поделити у две групе: рад на основном софтверу за потребе службе за студентске послове, и рад на софтверу за размену података са инф.системом универзитета.

Олакшан је рад корисницима програма захваљујући богатом графичком корисничком интерфејсу а повећана је ефикасност. Од јануара 2014. апликација се интензивно користи свакодневно. Дефинисан је и уведен у употребу интерни XMF формат података и екстензија за фајлове за унос оцена. Само основна десктоп апликација ЗЕУС садржи преко 60.000 линија кода који су написани за потребе факултета и овог пројекта у програмском језику C#.

5.3. Активности експерименталног истраживања

5.3.1. Планирање истраживања

Информациони систем и софтвер који је обухваћен у истраживању са циљем провере предложене методологије је у основи датацентричан информациони систем. Основни функционални захтеви испуњени кроз развој софтвера су усмерени на вођење потребних евиденција и израду различитих извештаја. Обраде података у оквиру процеса овог система по правилу нису временски критичне, није у питању систем за рад у реалном времену. Али јесте систем са комплексним обрадама података који ангажује значајне ресурсе.

Посматрани софтвер је у тренутку спровођења овог истраживања био у продукцији и пуној употреби већ две године. Из перспективе развоја софтвера то је фаза одржавања. Приликом пројектовања и развоја у основи је била парадигма протока података и примењена је Структурна системска анализа за моделовање система. Коришћен је итеративни и инкрементални принцип развоја. За потребе овог истраживања планирано је коришћење модела система развијеног применом ССА у претходним фазама развоја софтвера.

Планирано је дефинисање индикатора ефеката протока података, у даљем тексту индикатори, за сваки примитивни процес посматраног дела софтвера и имплементирање механизама надгледања базе података. По обављеним потребним дорадама система и након периода прикупљања података о раду система, планирано је спровођење анализе грешака софтвера у складу са описаном методологијом. Резултати спроведене анализе треба да одреде грешке које треба третирати као и редослед њиховог решавања.

План овог истраживања садржао је следеће фазе:

- Структурна системска анализа, провера и дорада
- Одабир процеса за проверу методологије

- Дефинисање индикатора за мерење протока података
- Имплементирање механизма надгледања у контролном периоду
- FMEA по предложеној методологији
- Одређивање и примена мера за унапређење система
- Надгледање система након примене мера
- Упоређење и анализа података пре и после унапређења система

5.3.2. Реализација структурне системске анализе

У току развоја софтвера спроведена је детаљна ССА комплетног система. ССА посматра систем као функцију, односно процес обраде података. Процес на основу улазних података генерише излазне податке, може се зато посматрати и као функција.

Приликом моделовања овог информационог система примењен је приступ који реалне потребе корисника препознаје и разликује од екстерних поступака који су у функцији остваривања реалних, есенцијалних потреба како то називају MacMenamin и Palmer у свом раду из 1984. године [88] где се такође потенцира и разлика између логичког и физичког модела система.

Циљ пројектовања и израде посматраног новог софтвера који је у основи овог истраживања био је да се на најбољи начин информационо подрже есенцијални процеси у реалном систему.

У оквиру спроведене ССА примењен је поступак моделирања процеса са формирањем модела који представља логичку, а не физичку спецификацију процеса, спецификација описује шта систем ради а не како ради. Описује шта је резултат процеса а не говори много или не говори ништа о начину имплементацији. У логичком моделу се што је више могуће занемарује физички аспект система. Оваквим приступом сваки дијаграм тока података креиран у оквиру ове анализе приказује суштину конкретног процеса занемаријући његову појавну форму. Каснија имплементација логичког модела подразумева креирање физичког дизајна система у датом технолошком и организационом окружењу. Приликом развоја софтвера сваки ток података из ССА је посматран као ток кроз који подаци теку на различитим носиоцима - медијима, као што су папирни материјали, датотеке пристигле преко компјутерске мреже, низ алфанумеричких знакова унетих од стране људског оператера преко тастатуре рачунара, и слично. Спецификација процеса међутим, ослобођена је имплементационих детаља као што је информација који је медијум носилац тока података.

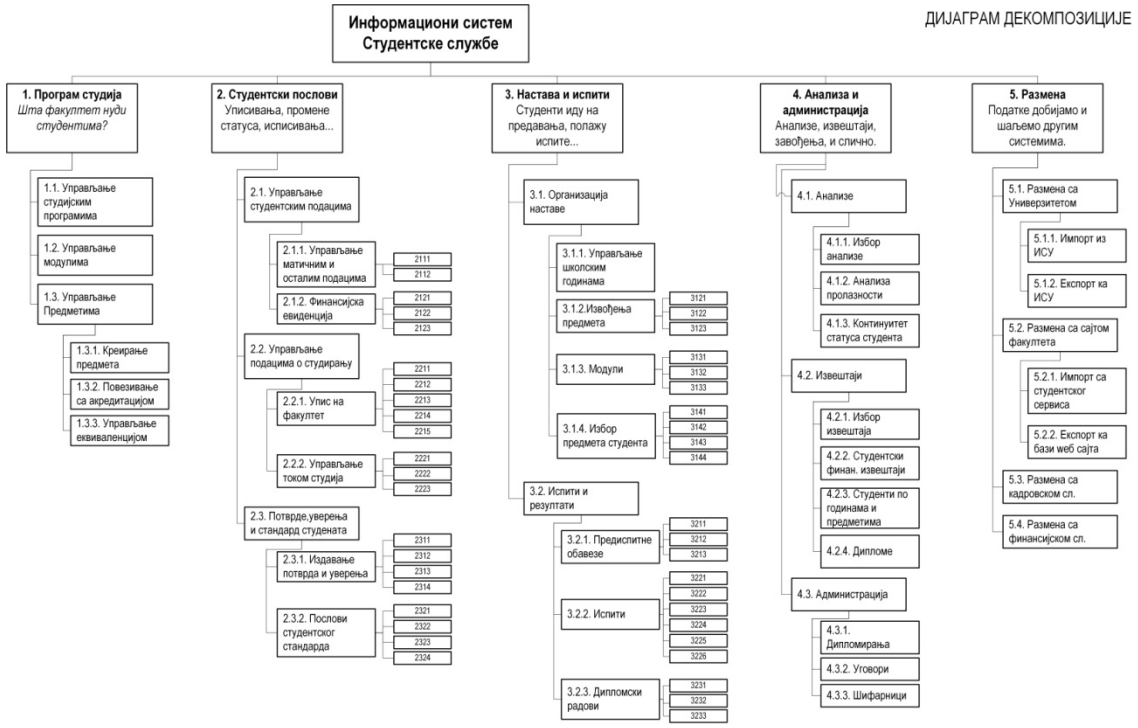
Моделирање процеса у посматраном информационом систему показало се као креативна активност. Као и током сваког моделирања у науци и инжењерским дисциплинама, кроз модел се представља знање о систему који се моделира. ССА користи као метод примену концепта модела. Да би се ова активност успешно спроводила потребно је познавање метода и техника које се користе. Искуство аналитичара, комуникација са стејкхолдерима и директним корисницима система, али пре свега познавање реалног система, представљају факторе од којих зависи квалитет креираних модела. Креирани логички модел информациопног система, као резултат примењене ССА, настао је као комбинација анализе и реверзибилног инжењеринга постојећег решења и познавања суштинских процеса реалног система.

Дијаграм контекста приказан на слици 21, приказује посматрани систем на највишем нивоу апстракције у моделу. Садржи представу система као процеса на врху хијерархије без икаквих интерних детаља и приказује његову интеракцију са спољним окружењем. Показује границе система који се анализира.

Слика 21: Дијаграм контекста

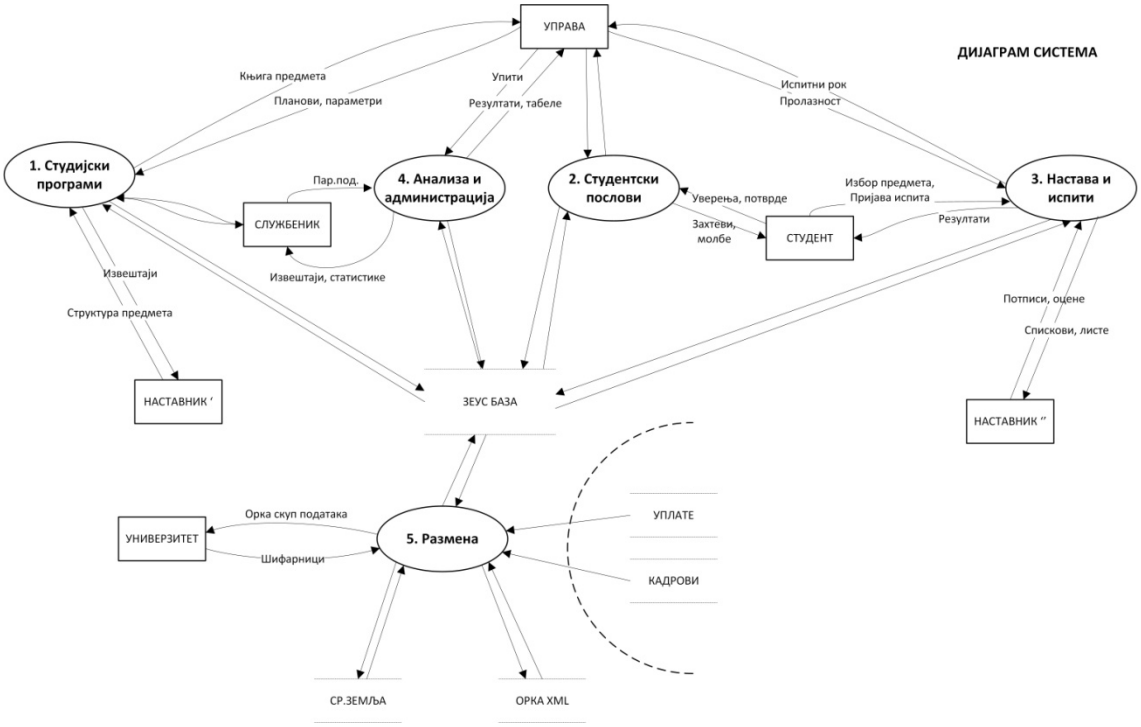
Приказани спољни објекти - интерфејси, су објекти који нису део посматраног система. Они се налазе ван посматраног информационог система студентске службе. То си екстерни објекти са којима систем комуницира.

Дијаграм контекста је декомпонован на различите дијаграме тока података у више нивоа и кроз више итерација. Број дијаграма у комплетном стаблу дијаграма је 94. Распоређени су у 4 нивоа и пет грана које представљају пет основних процеса, процеса првог нивоа. Комплетан дијаграм декомпозиције приказан је на слици 22, при чему су графички елементи за дијаграме четвртог нивоа приказани само бројем.



Слика 22: Дијаграм декомпозиције

Дијаграм система, или дијаграм нултог нивоа приказан је на слици 23. Он представља први дијаграм тока података у хијерархији - стаблу дијаграма, и приказује на унутрашњу структуру система на високом нивоу апстракције.



Слика 23: Дијаграм система

Идентификовано је пет процеса првог нивоа. То су:

1. Студијски програми

2. Студентски послови

3. Настава и испити

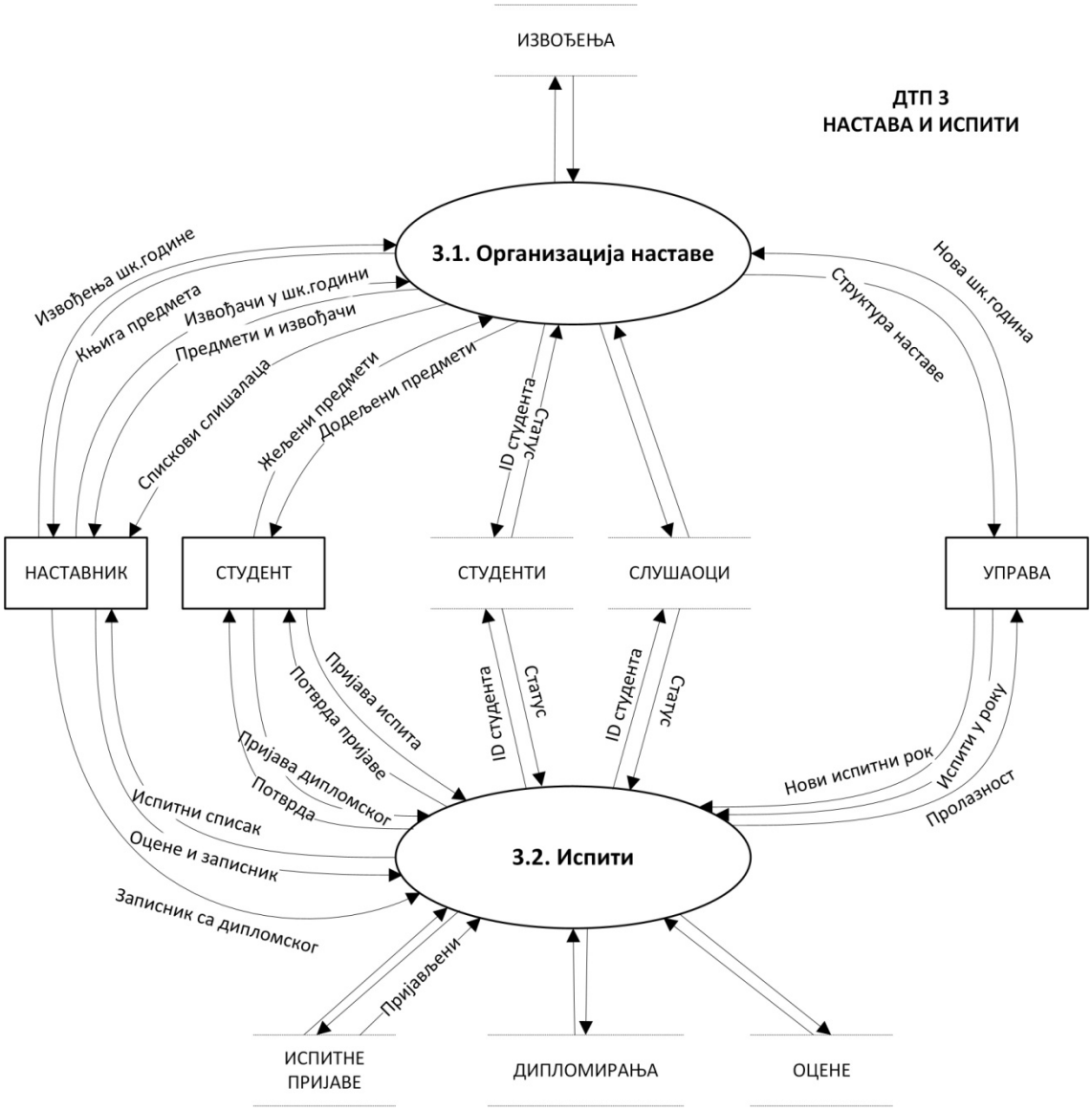
4. Анализа и администрација

5. Размена

Ови процеси представљени су даље припадајућим дијаграмима првог нивоа. Они представљају декомпозицију дијаграма система, односно дијаграма нултог нивоа.

Процес број 3 - "Настава и испити" је одабран за тестирање методологије и на њему је примењена методологија у свим појединостима. Приказан је на слици 24. Предложена методологија је тестирана у реалном окружењу и са реалним подацима на Машинском факултету Универзитета у Београду кроз суштинске

активности које се обављају и на другим факултетима. Ове активности су: похађања предавања из предмета који се налазе у књизи предмета, пријава и полагање испита, спровођење планираних студентских обавеза као и активност припреме и одбране дипломских радова и дипломирања. Овај скуп активности представља по мишљењу корисника система и аутора софтвера, најважнији део реалног система. Као такав је изабран за тестирање преложене методологије.



Слика 24: Дијаграм тока података "Настава и испити"

Списак процеса у оквиру процеса 3:

3.	Настава и испити
3.1.	Организација наставе
3.1.1.	Управљање школским годинама
3.1.2.	Извођења предмета
3.1.2.1.	Креирање извођења предмета
3.1.2.2.	Одређивање извођача предмета
3.1.2.3.	Креирање блокова предмета
3.1.3.	Модули
3.1.3.1.	Креирање студијских модула
3.1.3.2.	Одређивање обавезних предмета модула
3.1.3.3.	Одређивање изборних предмета модула
3.1.4.	Избор предмета студента
3.1.4.1.	Прикупљање и обрада жеља студената
3.1.4.2.	Додељивање предмета на основу жеља
3.1.4.3.	Директна додела предмета студенту
3.1.4.4.	Извештавање о додељеним предметима
3.2.	Испити и резултати
3.2.1.	Предиспитне обавезе
3.2.1.1.	Унос потписа
3.2.1.2.	Одрицање од потписа
3.2.1.3.	Извештавање
3.2.2.	Испити
3.2.2.1.	Управљање испитним роковима
3.2.2.2.	Управљање испитима
3.2.2.3.	Пријава испита

3.2.2.4. Формирање испитних спискова

3.2.2.5. Обрада резултата

3.2.2.6. Анализа испита

3.2.3. Дипломски радови

3.2.3.1. Пријава дипломског

3.2.3.2. Провера услова за дипломирање

3.2.3.3. Евидентирање дипломирања

Спроведена приказана ССА обезбедила је одгвоворе на питања: које функције поседује реални систем, односно процесе ИС, какав је однос између процеса, какве се трансформације података обављају у процесима, шта су улази а шта излази у процесима и одакле долазе и где одлазе подаци.

Декомпозиција процеса је реализована до нивоа примитивних процеса. Критеријум докле ће се ићи са даљом декомпозицијом процеса у оквиру ове ССА био је одговор на питање да ли посматрани процес садржи у себи суштински независне паралелне процесе. Декомпозиција је прекидана када би се дошло до природно секвенцијалних процеса које је могуће представити дијаграмом тока (дијаграмом тока програма) или псеудокодом. Као помоћни метод за одређивање да ли је у питању паралелни или секвенцијални процес, посматрано је коришћење суштинске меморије. Посебан значај идентификованих примитивних процеса за предложену методологију је у томе што се за примитивне процесе одређују индикатори ефеката протока података. Експерименталним истраживањем утврђено је да је у оквиру процеса 3 присутно 23 примитивна процеса. То су процеси са ознакама 3.1.1. , 3.1.2.1., 3.1.2.2., 3.1.2.3., 3.1.3.1., 3.1.3.2., 3.1.3.3., 3.1.4.1., 3.1.4.2., 3.1.4.3, 3.1.4.4., 3.2.1.1., 3.2.1.2., 3.2.1.3., 3.2.2.1., 3.2.2.2., 3.2.2.3., 3.2.2.4., 3.2.2.5., 3.2.2.6., 3.2.3.1., 3.2.3.2. и 3.2.3.3..

Процес 3 као један од пет сложених процеса ИС-а првог нивоа садржи процесе дефинисања и организације наставе, као и активности за подршку испитивања и оцењивања обављених обавеза студената. Састоји се од два

подпроцеса: "Организација наставе" и "Испити и резултати". Речник података овог процеса садржи опис и структуру одговарајућих складишта података што је приказано у табели 10.

Табела 10: Складишта података из процеса 3

Školske_godine <Godina, Opis, Status>		
Polje	*Tip*	*Ograničenje*
Godina	integer	NotNull
Opis	char(10)	NotNull
Status	bool	
Moduli <Naziv, Šifra, TipStudija, Max, Opis>		
Polje	*Tip*	*Ograničenje*
Naziv	char(10)	
Šifra	char(10)	NotNull
TipStudija	integer	in (210,220,320)
Max	integer	
Opis	char(128)	
Izvodjenja <Predmet, Šifra, Izborni, BlokPredmet, BlokŠifra, BlokUloga>		
Polje	*Tip*	*Ograničenje*
Predmet	integer	
Šifra	integer	
Izborni	bool	NotNull
BlokPredmet	integer	
BlokŠifra	integer	
BlokUloga	bool	
Studenti <ID, GodinaUpisa, RedniBrojRegistra, Kandidat, Ime, Prezime, ImeRoditelja, /JMBG, Pasos/, Drzavljanstvo, TipStudija, Email, Nacionalnost, PosebanUslov>		
Polje	*Tip*	*Ograničenje*
ID	integer	
GodinaUpisa	integer	NotNull
RedniBrojRegistra	char(10)	NotNull
Kandidat	integer	
Ime	char(128)	
Prezime	char(128)	

ImeRoditelja	char(128)	
JMBG	char(13)	lenght=13
Pasos	char(10)	
Drzavljanstvo	integer	
TipStudija	integer	in (210,220,320)
Email	char(128)	
Nacionalnost	integer	
PosebanUslov	integer	
PredmetiStudenta <Predmet, Izvodjenja, Student, Ocena, Stanje, Validan>		
Polje	*Tip*	*Ograničenje*
Predmet	integer	
Izvodjenje	integer	
Student	integer	
Ocena	integer	in (5,6,7,8,9,10)
Stanje	integer	
Validan	bool	

Увидом у спроведену ССА установљено је да су идентификовани основни процеси, да је реализована декомпозиција система, креирани дијаграми тока података и дијаграми тока алгоритама примитивних процеса. Декомпоновани су токови података и складишта података. Написани су речници података који описују садржај и структуру токова и складишта података. Урађена је Спецификација логике примитивних процеса.

5.3.3. Одређивање индикатора ефеката протока података

Сви наведени примитивни процеси обухваћени су процесом одређивања индикатора. Индикатори су одређени из перспективе протока података. Посматрано на нивоу физичког модела, односно имплементације, посматрани процеси се по природи обавезно завршавају или уписом неког податка у базу података или испоруком неког податка директно из базе или обрађеног ка кориснику система или комбинацијом ових догађаја.

Дефинисани и коришћени индикатори су финални индикатори или индикатори резултата. За неке процесе су у току овог истраживања осмишљени и прелазни индикатори који ће се користити за праћење стања процеса у специфичним ситуацијама и обрадама које су повезане са екстерним информационим системима. Овим истраживањем сви дефинисани индикатори су одређени као квантитативни индикатори. При дефинисањеу индикатора, водило се рачуна о планираној имплементацији са што мањим ангажовањем ресурса.

Функционисање информационог система факултета прати ритам реалних активности на факултету. Активности се обављају у циклусима који представљају школске године. Школске године покривају низ фаза и активности које се понављају сваке године. Почетак и крај школске године се не поклапа са календарском годином, али период на који се односе активности везане за једну школску годину је приближан трајању једне године. Неке активности које припадају једној школској години могу се дешавати истовремено док трају активности које припадају другој школској години. Осим овог примарног годишњег циклуса активности, постоје и други циклуси мањег трајања и обично су везани за организацију предавања и испитне рокове. Већина дефинисаних индикатора је везана за циклус школске године.

Финални индикатори, који се генерално морају сматрати као најважнији, у нашем случају омогућавају оцену резултата-ефеката процеса који могу бити позитивни и негативни. Неки од дефинисаних индикатора мере позитивне ефекте и имају назовимо то "позитивну логику", а неки погодније изражавају негативне

ефекте и имају "негативну логику", што је важно за поређење вредности индикатора у различитим периодима.

Као резултат ове фазе истраживања, дефинисани су потребни индикатори, одређена су им индикативна и дескриптивна имена, као и опис шта они тачно представљају. Индикатори су тако одређени да изражавају укупан и коначан резултат обраде у процесу.

У даљем истраживању анализирани су финални индикатори са циљем да се одреде ефекти протока података који произлазе из сваког појединачног процеса.

Дефинисани су следећи индикатори:

Процес 3.1.1. Организација наставе / Управљање школским годинама.
Индикатор 3.1.1.-1 Број успешно обављених ажурирања
Посматра се једна школска година. Позитивна логика.

Процес 3.1.2.1. Организација наставе / Извођења предмета / Креирање извођења предмета.
Индикатор 3.1.2.1-1 Број креираних извођења предмета.
Индикатор 3.1.2.1-2 Број накнадно креираних извођења предмета.
Посматра се једна школска година. Позитивна логика.
Индикатор 3.1.2.1-3 Број ажурирања-исправки креираних извођења предмета.
Посматра се једна школска година. Негативна логика.

Процес 3.1.2.2. Организација наставе / Извођења предмета / Одређивање извођача предмета.
Индикатор 3.1.2.2.-1 Број унетих извођача предмета за школску годину.

Индикатор 3.1.2.2.-2 Број унетих нових извођача предмета за школску годину.
Посматра се једна школска година. Позитивна логика.
Индикатор 3.1.2.2.-3 Број ажурирања-исправки извођача предмета за школску годину.
Посматра се једна школска година. Негативна логика.

Процес 3.1.2.3. Организација наставе / Извођења предмета / Креирање блокова предмета.
Индикатор 3.1.2.3.-1 Број повезаних блокова предмета.
Посматра се једна школска година. Позитивна логика.

Процес 3.1.3.1. Организација наставе / Модули / Креирање студијских модула.
Индикатор 3.1.3.1.-1 Број креираних модула у школској години.
Посматра се једна школска година. Позитивна логика.

Процес 3.1.3.2. Организација наставе / Модули / Одређивање обавезних предмета модула.
Индикатор 3.1.3.2.-1 Број унетих атрибута предмета модула.
Посматра се једна школска година. Позитивна логика.

Процес 3.1.3.3. Организација наставе / Модули / Одређивање изборних предмета модула.
Индикатор 3.1.3.3.-1 Број унетих атрибута предмета модула.
Посматра се једна школска година. Позитивна логика.

Процес 3.1.4.1. Организација наставе / Избор предмета студента / Прикупљање и обрада жеља студената.
Индикатор 3.1.4.1.-1 Број прикупљених захтева за доделу предмета.

Посматра се једна школска година. Позитивна логика.
Индикатор 3.1.4.1.-2 Број неисправних захтева за доделу предмета.
Посматра се једна школска година. Негативна логика.

Процес 3.1.4.2. Организација наставе / Избор предмета студента / Додељивање предмета на основу жеља.
Индикатор 3.1.4.2.-1 Број додељених предмета студенту.
Индикатор 3.1.4.2.-2 Број додељених предмета студенту у првом кругу обраде.
Индикатор 3.1.4.2.-3 Број додељених предмета студенту у другом кругу обраде.
Индикатор 3.1.4.2.-4 Број додељених предмета студенту у трећем кругу обраде.
Индикатор 3.1.4.2.-5 Однос броја додељених ЕСПБ-а и могућег броја ЕСПБ-а студента.
Посматра се једна школска година. Позитивна логика.

Процес 3.1.4.3. Организација наставе / Избор предмета студента / Директна додела предмета студенту.
Индикатор 3.1.4.3.-1 Број додељених предмета студенту.
Посматра се једна школска година. Позитивна логика.

Процес 3.1.4.4. Организација наставе / Избор предмета студента / Извештавање о додељеним предметима.
Индикатор 3.1.4.4.-1 Број генерисаних извештаја о предметима студента на захтев.
Индикатор 3.1.4.4.-2 Број индиректно генерисаних извештаја о предметима студента.
Посматра се једна школска година. Позитивна логика.
Индикатор 3.1.4.4.-3 Просечно време обраде података за потребе генерисања извештаја о предметима студента.
Посматра се једна школска година. Негативна логика.

Процес 3.2.1.1. Испити и резултати / Предиспитне обавезе / Унос потписа.
Индикатор 3.2.1.1.-1 Број евидентираних "потписа" - потврда о испуњеним предиспитним обавезама.
Индикатор 3.2.1.1.-2 Број евидентираних "потписа" који су уграђени у студентске финансијске обрачуне.
Посматра се једна школска година. Позитивна логика.

Процес 3.2.1.2. Испити и резултати / Предиспитне обавезе / Одрицање од потписа.
Индикатор 3.2.1.2.-1 Број евидентираних одрицања.
Посматра се једна школска година. Позитивна логика.

Процес 3.2.1.3. Испити и резултати / Предиспитне обавезе / Извештавање.
Индикатор 3.2.1.3.-1 Број издатих извештаја.
Посматра се једна школска година. Позитивна логика.
Индикатор 3.2.1.3.-2 Просечно време потребно за израду извештаја.
Индикатор 3.2.1.3.-3 Просечно време потребно за пропагацију података кроз цео систем укључујући и сајт факултета и Студентски сервис.
Посматра се једна школска година. Негативна логика.

Процес 3.2.2.1. Испити и резултати / Испити / Управљање испитним роковима.
Индикатор 3.2.2.1.-1 Број креираних испитних рокова на свим студијским програмима.
Посматра се једна школска година. Позитивна логика.
Индикатор 3.2.2.1.-2 Просечно време потребно за креирање испитног рока.
Индикатор 3.2.2.1.-3 Просечно време потребно за пропагацију података кроз цео систем укључујући и сајт факултета и Студентски сервис.
Посматра се једна школска година. Негативна логика.

Процес 3.2.2.2. Испити и резултати / Испити / Управљање испитима.
Индикатор 3.2.2.2.-1 Број креираних испита.

Индикатор 3.2.2.2.-2 Број креираних и одржаних испита.
Посматра се једна школска година и испитни рок. Позитивна логика.
Индикатор 3.2.2.2.-3 Просечно време потребно за креирање испита.
Индикатор 3.2.2.2.-4 Просечно време потребно за пропагацију података кроз цео систем укључујући и сајт факултета и Студентски сервис.
Посматра се једна школска година и испитни рок. Негативна логика.

Процес 3.2.2.3. Испити и резултати / Испити / Пријава испита.
Индикатор 3.2.2.3.-1 Број евидентираних пријава испита.
Индикатор 3.2.2.3.-2 Број евидентираних пријава испита са полагањем.
Посматра се једна школска година, испитни рок и испит. Позитивна логика.
Индикатор 3.2.2.3.-3 Просечно време потребно за евидентирање пријаве испита.
Индикатор 3.2.2.3.-4 Просечно време потребно за пропагацију података кроз цео систем укључујући и сајт факултета и Студентски сервис.
Посматра се једна школска година, испитни рок и испит. Негативна логика.

Процес 3.2.2.4. Испити и резултати / Испити / Формирање испитних спискова.
Индикатор 3.2.2.4.-1 Број формираних испитних спискова.
Индикатор 3.2.2.4.-2 Број појединачних записа испитних спискова.
Посматра се једна школска година и испитни рок. Позитивна логика.
Индикатор 3.2.2.4.-3 Број накнадно додатих појединачних записа испитних спискова.
Посматра се једна школска година и испитни рок. Негативна логика.

Процес 3.2.2.5. Испити и резултати / Испити / Обрада резултата.
Индикатор 3.2.2.5.-1 Број обрађених испитних спискова.
Индикатор 3.2.2.5.-2 Број појединачних записа обрађених испитних спискова.
Индикатор 3.2.2.5.-3 Број појединачних записа позитивних оцена обрађених испитних спискова.
Посматра се једна школска година, испитни рок и испит. Позитивна логика.

Индикатор 3.2.2.5.-4 Број неуспешно обрађених појединачних записа испитних спискова.
Посматра се једна школска година, испитни рок и испит. Негативна логика.

Процес 3.2.2.6. Испити и резултати / Испити / Анализа испита.
Индикатор 3.2.2.6.-1 Број генерисаних анализа.
Посматра се једна школска година. Позитивна логика.
Индикатор 3.2.2.6.-2 Однос времена утрошеног за генерисање анализа и броја појединачних записа.
Посматра се једна школска година. Негативна логика.

Процес 3.2.3.1. Испити и резултати / Дипломски радови / Пријава дипломског.
Индикатор 3.2.3.1.-1 Број пријављених дипломских радова.
Посматра се једна школска година. Позитивна логика.

Процес 3.2.3.2. Испити и резултати / Дипломски радови / Провера услова за дипломирање.
Индикатор 3.2.3.2.-1 Просечно време потребно за давање одобрења за израду.
Посматра се једна школска година. Негативна логика.

Процес 3.2.3.3. Испити и резултати / Дипломски радови / Евидентирање дипломирања.
Индикатор 3.2.3.3.-1 Број евидентирања дипломирања.
Посматра се једна школска година. Позитивна логика.
Индикатор 3.2.3.3.-2 Однос броја накнадног ажурирања и броја евидентирања дипломирања.
Посматра се једна школска година. Негативна логика.

Укупно је дефинисано 52 индикатора. Од тога 39 индикатора који мере учинак, а 13 индикатора који мере ефикасност, 35 индикатора са позитивном логиком (већи број значи бољи успех процеса) а 17 индикатора са негативном логиком (мањи број значи бољи успех процеса). 37 индикатора се посматра на нивоу периода једне школске године, а 15 индикатора се посматра и на мањем временском периоду.

5.3.4. Имплементација надгледања система

Сви посматрани процеси остварују интеракцију са базом података. Дефинисани су индикатори са циљем одређивања резултата процеса са становишта протока података. За потребе обезбеђивања података потребних за одређивање вредности индикатора спроведене су потребне измене у апликацији и у бази података.

Актуелно софтверско решење састављено је од више засебних апликација са коришћењем више различитих релационих система база података (MySQl, Oracle, PostgreSQL, MS SQL Server), са више засебних шема података. Основна шема података налази се на систему са слабом подршком за надгледање (auditing) и где покретање снимања general query log-a троши ресурсе до нивоа и до повремено 20% детектованог успорења рада. Имајући све то у виду, у оквиру овог истраживања коришћена је комбинација употребе окидача за једноставне процесе и апликативног приступа за све остале процесе за потребе обезбеђивања података за дефинисане индикаторе.

За све посматране процесе који као свој резултат имају промену стања неког ентитета у бази података, спроведена је провера имплементације препорученог сценарија обраде података са чувањем историје промена пре него само новог стања. Овакав начин организације података обезбеђује додатне информације о ентитету, не само какво је тренутно стање већ и како је дошло до њега, обично и ко је и када обавио измене. У зависности од обима историјских података, сложености израчунавања стања и захтеване брзине одзива, може се одржавати паралелно са историјом и стање, сума, конфигурација ентитета, али то објективно представља редундансу што има своје лоше последице.

Активности праћења и меморисања активности у посматраном информационом систему по предложеној методологији реализоване су у два дела. Један је био модификација базе података са променама у начину организације података, логичком и физичком моделу. Примењени модели и поступци су

компатибилни и са већином познатих SQL база података. Други део посла односио се на увођење нових функционалности софтверске апликације.

Најпре је креирана нова структура података и имплементирана издвојена шема података у бази, за смештање података потребних за индикаторе. Упис података у ову издвојену структуру обезбедио је неометани рад основне базе података.

У зависности од природе процеса спроведене су различите модификације система.

- За једноставне процесе који као резултат имају промену података најчешће једног ентитета у бази података, код којих је смер протока података ка складишту података, коришћени су окидачи као најједноставнији и довољан механизам. Раније је проверено да сви па и једноставни процеси имплементирају модел чувања историје промена.
  На пример Процес 3.2.3.3. Испити и резултати / Дипломски радови / Евидентирање дипломирања, представља једноставан процес за који је креирана табела за потребе надгледања:
  CREATE TABLE `diplomiranja_audit`(
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `diplomiranje` int(11) NOT NULL,
  `skolskagodina` int(11) NOT NULL,
  `promena` varchar(20) COLLATE utf8_bin DEFAULT NULL,
  `vreme` datetime DEFAULT NULL, `operater` int(11) NOT NULL,
  PRIMARY KEY(`id`));

  и постављен је окидач на одговарајућој табели:
  CREATE TRIGGER `posle_diplomiranja_audit`
  AFTER INSERT
  ON diplomski
  FOR EACH ROW
  INSERT INTO diplomiranja_audit

```
SET promena = 'novo',
vreme = NOW(),
diplomiranje = NEW.iddiplomski,
skolskagodina = NEW.skolskagodina,
operater = NEW.idoperater;
```

- За све сложеније процесе обраде података где је укључено више ентитета из базе података које сматрамо сложенијим без обзира на секвенцијалност логике примитивног процеса, примењена је апликативна контрола евидентирања реализације процеса. Различите операције са табелама у базама података у оквиру ових процеса, укључујући сада и операције евидентирања самог процеса, повезане су у трансакције како би се обезбедила конзистентност базе података. Завршетак трансакције је увек операција уписа података о обављеном процесу у ентитет посебне шеме за евиденцију тог процеса. На пример Процес 3.1.4.3. Организација наставе / Избор предмета студента / Директна додела предмета студенту, имплементиран је, поједностављено приказано, следећим C# кодом:

```
MySqlTransaction tr = null;
try
{
        tr = veza.BeginTransaction();
        MySqlCommand cmd = new MySqlCommand();
        cmd.Connection = veza;
        cmd.Transaction = tr;
        cmd.CommandText =komandaObrada1; cmd.ExecuteNonQuery();
        cmd.CommandText =komandaObrada2; cmd.ExecuteNonQuery();
        cmd.CommandText =komandaAudit; cmd.ExecuteNonQuery();
        tr.Commit();
        Klase.PorukaOK();
}
catch (MySqlException ex1)
```

```
{
        try { tr.Rollback(); }
        catch (MySqlException ex2)
        { Klase.PorukaGreska(ex2);  }
        Klase.PorukaGreska(ex1);
}
```

- Уведени су додатни подаци у модел података за све процесе где се индикаторима прати ефикасност процеса а не само укупан учинак и где је време трајања процеса део индикатора. Ови процеси се третирају као сложени и све наведено за такве процесе је спроведено и за ове процесе.

- Процеси који су усмерени са протоком података ка интерфејсу од базе података, као што су разни процеси извештавања, прегледа, читања података и информација из базе података, захтевали су нове елементе у моделу података. Као што RDBMS не обезбеђује окидач за select команду, тако и већина софтвера који раде са базама података, по правилу не меморишу информацију о читању података. За све такве процесе у скупу посматраних процеса, допуњен је модел података и креирани су елементи у бази података са циљем евидентирања и таквих процеса. Осим дневника процеса корисно би било и трајно чување формираних издатих излазних сетова података као архиве копија излаза из система.

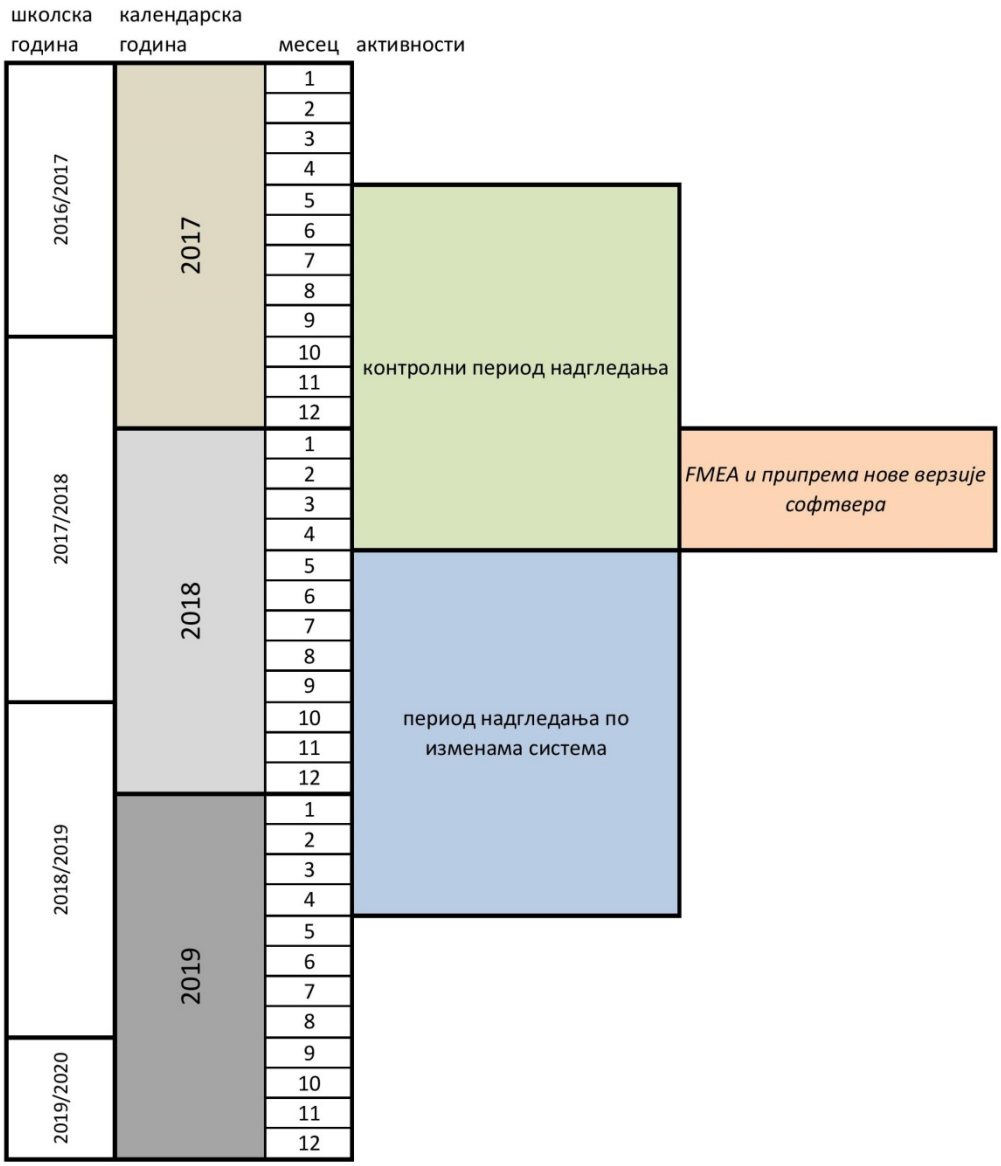
Сам поступак праћења и мерења индикатора перформанси процеса одвијао се најпре у времену од годину дана, а добијени резултати послужили су као основа за унапређење и управљање перформансама и процесима. Добијене информације су употребљене за поређење са резултатима из претходних мерења.

5.3.5. Надгледање система у контролном периоду

Надгледање информационог система Студентске службе на начин како је предвиђено предложеном методологијом започето је од 1.1.2017. године са једним мањим бројем индикатора. Надгледања система за све индикаторе предвиђене овим истраживањем омогућено је и започето од 1.5.2017. године.

Кључне активности на факултету се реализују у истим понављајућим циклусима, где један циклус траје годину дана. Те активности могу бити груписане у календарску, или чешће школску годину што зависи од перспективе посматрача, али суштински исте важне активности се понављају стално са периодом понављања од годину дана. Неке од најважнијих активности су: конкурс за упис на факултет, пријемни испит, упис на факултет, упис у школску годину, додела предмета за школску годину студентима, дефинисање списка предмета за школску годину, одређивање предавача, одржавање испитних рокова и слично. Активности овако поређане, са уписом нове генерације студената на факултет, јесте перспектива која је изабрана за редослед и груписање активности у годишњи циклус активности за потребе надгледања система и одређивање индикатора процеса за два узастопна циклуса рада информационог система.

Пред крај контролног периода надгледања, у периоду јануар-април 2018. године, спроведена је FMEA онако како је предвиђено методологијом, и изабране мере за отклањање грешака, односно унапређење информационог система, имплементиране су највећим делом кроз модификације и дораде софтвера система. Нова верзија софтвера уведена је у рад 1.5.2018. године, и од тада је започео и нови период надгледања система и прикупљања података за дефинисане индикаторе, као и нови годишњи циклус активности на факултету. Календар спроведених активности је приказан на слици 25 и показује када су прикупљени подаци за израчунавање индикатора из почетног-контролног периода и периода рада рада измењеног система.



Слика 25: Календар активности надгледања система

Прикупљање потребних података спроведено је на организован, систематичан, системски и аутоматски начин. Једном уведен систем, није захтевао накнадни рад на развоју, осим неких проширења скупа индикатора који су ван овог истраживања а послужили су за унакрсну проверу сумарних параметара рада система посебно везано за размене података са екстерним

системима. У табели 11 приказане су вредности индикатора на основу прикупљених података у контролном периоду.

Табела 11:Вредности индикатора за контролни период

Процес	Индикатор	Вредност
3.1.1.	1	5
3.1.2.1.	1	1500
3.1.2.1.	2	72
3.1.2.1.	3	21
3.1.2.2.	1	4952
3.1.2.2.	2	390
3.1.2.2.	3	45
3.1.2.3.	1	185
3.1.3.1.	1	24
3.1.3.2.	1	750
3.1.3.3.	1	2254
3.1.4.1.	1	26080
3.1.4.1.	2	1860
3.1.4.2.	1	19990
3.1.4.2.	2	12000
3.1.4.2.	3	6000
3.1.4.2.	4	1990
3.1.4.2.	5	0,623
3.1.4.3.	1	2650
3.1.4.4.	1	2200
3.1.4.4.	2	5000
3.1.4.4.	3	1,24
3.2.1.1.	1	17500
3.2.1.1.	2	17150
3.2.1.2.	1	50
3.2.1.3.	1	1500
3.2.1.3.	2	1,33
3.2.1.3.	3	2,5
3.2.2.1.	1	30
3.2.2.1.	2	145,5
3.2.2.1.	3	2,2
3.2.2.2.	1	25000

3.2.2.2.	2	20000
3.2.2.2.	3	0,02
3.2.2.2.	4	3,8
3.2.2.3.	1	295000
3.2.2.3.	2	180000
3.2.2.3.	3	1,56
3.2.2.3.	4	3,7
3.2.2.4.	1	24990
3.2.2.4.	2	289000
3.2.2.4.	3	13000
3.2.2.5.	1	24985
3.2.2.5.	2	288842
3.2.2.5.	3	102555
3.2.2.5.	4	275
3.2.2.6.	1	6850
3.2.2.6.	2	1,1
3.2.3.1.	1	750
3.2.3.2.	1	45,5
3.2.3.3.	1	742
3.2.3.3.	2	0,085

5.3.6. Примена FMEA методе

Као централни део предложене методологије, спроведена је у овом истраживању FMEA анализа грешака посматраног дела информационог система "Настава и испити", са фокусом на апликативни софтвер.

Сви кораци и све активности предвиђене методологијом су доследно спроведене. Идентификоване су грешке и проблеми у раду информационог система везано за посматрани подсистем. Коришћене су дефинисане лингвистичке скале за оцењивање фактора ризика. Релативни значај фактора ризика одређен је узимајући у обзир идентификоване грешке. Ранг грешака је добијен применим предложене фази COPRAS методе. Користећи резултат спроведене анализе у облику ранг листе грешака по свом значају и одређених мера за превазилажење тих грешака, приступило се модификацијама софтвера. Нова оперативна верзија софтвера, као и дефинисане и са стејкхолдерима договорене измене у процедурама, уведени су у рад од 1.5.2018. године.

Анализа грешака физичких система узима у обзир као фактор ризика старење или пропадање материјала и конструкција, али софтверски модули не пропадају на тај начин. Мада софтверски производи имају ефекте застаревања, то не утиче на начине кварења и појаву грешака као код материјалних система. Софтвер је неопипљив, његове грешке зависе од динамичког понашања система. Због тога је идентификовање грешака и начина испољавања грешака софтвера, сложен процес.

У току овог истраживања развијене су и интензивно коришћене прве верзије CASE софтвера за подршку методологији.

5.3.6.1. Идентификовање грешака

Основни фокус при идентификовању грешака био је на грешкама у функционисању имплементираних функционалности софтвера. У фази одржавања софтвера, програмери праве измене које траже корисници да би модификовали систем и подржали промењене услове пословања. Те промене су неопходне да би систем остао у употреби и да би се успешније и дуже користио. На тај начин посматрано, одржавање није одвојена посебна фаза у животном циклусу софтвера, већ је то понављање других фаза животног циклуса потребних да се потребне промене проуче и имплементирају. Количина времена и ресурса које је потребно уложити у одржавање софтвера великим делом зависи од перформанси претходних фаза животног циклуса. Анализа и решавање евентуално преосталих нерешених почетних функционалних захтева као и дефинисање нових захтева због промена окружења, нису предмет ове методологије. Проширење примењене методологије такође могло би да предвиди да се у овакву FMEA за софтверски засноване системе укључе и хардверске грешке.

Идентификована је 41 грешка у оквиру посматраног дела система. Све грешке су повезане са процесима из спроведене ССА. Одређени су узроци који доводе до настанка грешака као и последице које настају материлазицијом истих. Списак грешака приказан је у табели 12.

Табела 12: Индетификоване грешке

Р.б.	**Процес**	**Грешка**	**Последица**	**Узрок**
1.	Креирање извођења предмета	Креирање извођења погрешног предмета	Погрешан унос података и поступак исправке и поништавања грешке	Кориснички интерфејс за табеларни приказ није довољно флексибилан
2.	Креирање извођења предмета	Креирање извођења погрешне верзије предмета	Погрешан предмет у књизи предмета, неисправна додела предмета студентима.	Грешка у импорту података из спољне евиденције предмета

3.	Креирање извођења предмета	Унет неодговарајући семестар извођења	Предмет се налази на погрешној позицији. Погрешан избор и додела предмета студентима	Комплексан скуп података у кор. интерфејсу књиге предмета
4.	Одређивање извођача предмета	Није унет извођач предмета.	Одсуство информације о предавачима у школској години, може бити немогућ унос оцене на испиту.	Неажуран списак наставника
5.	Одређивање извођача предмета	Погрешан редослед списка извођача.	Одсуство информација о предметима у школској години важних за студенте.	Редослед извођача предмета имплицитан.
6.	Креирање студијских модула	Неисправна CQЛ команда за креирање модула	Погрешан унос података, исправке и поништавања грешке.	Креирање модула ван кор. интерфејса
7.	Одређивање обавезних предмета модула	Није креирано извођење предмета за одређени модул	Структура предмета на модулу није комлетна, студенти немају комплетну информацију.	Креирање обавезних предмета модула методом вишеструких извођења
8.	Одређивање изборних предмета модула	Изборни предмет није унет на предвиђеном редном месту.	Погрешан редослед предмета за избор на одређеној позицији. Број слушалаца предмета не одговара важности предмета на модулу.	Изборни предмети нису сортирани.
9.	Одређивање изборних предмета модула	Структура унакрсне табеле није одговарајућа.	Погрешан списка предмета модула.	Креирање списка предмета ван кор. интерфејса коришћењем унакрсне табеле.

10.	Прикупљање и обрада жеља студената	Погрешно прорачунат ранг студента на основу оцена.	Погрешна позиција на листи слушалаца предмета, студенту може бити недодељен предмет.	Неажурност оцена студената. Непроверене оцене у евиденцији.
11.	Прикупљање и обрада жеља студената	Нерешене жеље за предметима студената	Студенту нису додељени сви предмети у оквиру броја ЕСПБ.	Некомпатибилан скуп услова и ограничења за изборне предмете.
12.	Прикупљање и обрада жеља студената	Некомплетан скуп захтева студента при избору предмета.	Студенту неће бити обрађене жеље делимично или у потпуности.	Кор. интерфејс делимично спречава унос неисправних података.
13.	Прикупљање и обрада жеља студената	Нису доступни подаци о студентским жељама по предметима.	Студент нема информацију о рангу предмета, наставник нема информацију о тренду.	Статистика исказаних жеља се приказује из екстерног система
14.	Прикупљање и обрада жеља студената	Нису доступни ажурни подаци о студентским жељама по предметима.	Одлуке за обраду изборних жеља засноване су на неажурним подацима.	Дневна ажурност података.
15.	Додељивање предмета на основу жеља	Успорен рад система	Пад ефикасности рада корисника система у току обраде.	Масовна батцх обрада захтева.
16.	Директна додела предмета студенту	Доделу предмета студенту потребно је понављати.	Студенту нису додељени сви предмети са листе.	Кор. интерфејс предвиђа доделу појединачног предмета
17.	Директна додела предмета студенту	Додела предмета који је попуњен или није активан у текућој школској години.	Студенту је додељен предмет који није исправан. Неисправан списак слушалаца предмета.	Интерфејс не приказује попуњеност предмета

18.	Извештавање о додељеним предметима	Спискови су неажурни.	Неисправни подаци који су потребни наставницима и управи.	Генерисање извештаја о слушаоцима предмета ван система.
19.	Унос потписа	Потпис стечен у текућој години је третиран као раније стечен потпис.	Неисправно обрачуната школарина студента за текућу школску годину.	Непрецизна процедура, непоштовање процедуре.
20.	Унос потписа	Унет је потпис из предмета са новом шифром.	Финансијско задужење студента неће бити исправно, нетачан списак за слушање предмета.	Непостојање јединственог шифарника предмета факултета.
21.	Одрицање од потписа	Одрицање од потписа није обрађено.	Студент неће бити у могућности да присуствује настави.	Укидање потписа кроз наставнички сервис има свој временски прозор.
22.	Извештавање	Спискови по предметима и спискови по сменама нису компатибилни.	Студент неће бити у могућности да присуствује настави у смени.	Није прецизно дефинисан појам смене, корисници нису упућени.
23.	Управљање испитима	Креиран испит са неисправним параметрима.	Евидентиран положени испит који није био активан одређене школске године.	Нема провере године испита и извођења предмета.
24.	Пријава испита	Пријава испита сврстана у погрешну категорију без наплате.	Финансијски губитак за факултет.	Нетачан број ранијих полагања испита због увођења нове шифре.
25.	Пријава испита	Испит је пријављен више пута за исти испитни рок.	Немогућ унос свих оцена за конкретан испит.	Постојање два засебна поступка пријаве испита.

26.	Пријава испита	Пријава испита свтстана у погрешну категорију са наплатом.	Неоправдано финансијско задуживање студента.	Нетачан број ранијих полагања испита због блок рокова.
27.	Формирање испитних спискова	Листа пријављених без датума испита	Немогућ унос оцена.	Недефинисан датум испита у фајлу са испитним пријавама.
28.	Формирање испитних спискова	Листа пријављених са дуплираним студентом	Немогућ унос оцена.	Два засебна поступка пријаве испита.
29.	Обрада резултата	Неисправан број индекса.	Немогућа обрада оцене, нетачна статистика испита.	Унос броја индекса студента без провере.
30.	Обрада резултата	Неприхватљив унос оцене.	Немогућа обрада оцене, нетачна статистика испита.	Унос пријаве без провере предмета.
31.	Обрада резултата	Неисправан хмф фајл са оценама	Немогућа обрада оцена, нетачна статистика испита.	Непоштовање процедура, незаштићен и оффлине режим рада.
32.	Обрада резултата	Некомпатибилан хмф фајл са оценама	Немогућа обрада оцена, нетачна статистика испита.	Непоштовање процедура, погрешна верзија Зап програма.
33.	Обрада резултата	Компромитован садржај хмф фајла са оценама.	Немогућа обрада оцена, нетачна статистика испита.	Озбиљно непоштовање процедура, неправилно уређивање хмф фајла.
34.	Обрада резултата	Неприхватљив унос оцене студента	Немогућа обрада оцена, нетачна статистика испита.	Унос пријаве без провере статуса студента.

35.	Анализа испита	Неисправна анализа пролазности на испиту	Одлуке везане за пролазност засноване су на нетачним подацима.	Неправилно евидентирање студената на испиту.
36.	Провера услова за дипломирање	Нетачан број ЕСПБ-а студента.	Неефикасан процес провере услова.	Извештај не садржи потребне податке, подаци могу бити непроверени.
37.	Провера услова за дипломирање	Нетачан број положених испита студента.	Неефикасан процес провере услова.	Извештај не садржи потребне податке, подаци могу бити непроверени.
38.	Евидентирање дипломирања	Несагласност просечне оцене у унетим подацима.	Нетачна просечна оцена у дипломи.	Просечна оцена се евидентира директним уношењем.
39.	Евидентирање дипломирања	Датум дипломирања није последњи датум испита.	Редослед испуњених обавеза студента неће бити у складу са правилима студијског програма.	Непоштовање процедура наставника завршних предмета.
40.	Евидентирање дипломирања	Чланови комисије нетачно поређани	Подаци о члановима комисије на дипломи неће одговарати записнику са дипломирања.	Одређивање редног броја члана комисије редоследом уношења.
41.	Евидентирање дипломирања	Завршни испит на студијском програму није последњи положени	Израда дипломе неће бити могућа.	Непоштовање процедура наставника при уносу оцена.

У овом процесу идентификовања грешака, од посебне користи је било вертикално повезивање елемената логичког дизајна са начинима имплементације и искуствима из фазе одржавања. Осим искуства и знања аутора софтвера коришћено је доступно искуство из праксе развоја сличних софтвера. Уобичајени

начини настанка грешака код оваквог софтвера су грешке интерфејса, редослед и логика обављања операција обраде, грешке складишта података, некомпатибилност подсистема и слично. Сви ови из праксе познати типови грешака су узети у разматрање. У анализи је коришћено знање о софтверу и примењеним технологијама при имплементацији. У току фазе одржавања посматраног софтвера константно се прате и евидентирају грешке у раду и сви евентуални поремећаји и прекиди. Одржавање система обавља исти тим који се бавио и развојем. Када информациони систем ради у некој организацији, осим софтверских професионалаца од великог значаја су и крајњи корисници који често проналазе проблеме везане за његов рад и предлажу боље начине извршавања његових функција. Превазилажење проблема у разумевању између корисника и софтверских инжењера је веома важно. На слици 26, приказан је уобичајени вишекорисnички паралелни рад корисника софтвера у студентској служби факултета.

Током идентификовања грешака коришћени су подаци о насталим грешкама из поретходног периода записаним у дневник одржавања. Дневник одржавања у оквиру посматраног софтверског пројекта, представља евиденцију важних догађаја везаних за експлоатацију софтвера као што су: увођење нових верзија, крупне off-line обраде података, сервисне интервенције на базама података, крупни хардверски захвати а посебно све везано за сервере. У дневнику су детаљно евидентирани прекиди у раду, пријављене и детектоване апликативне грешке или други поремећаји у раду. Неке софтверске грешке идентификоване су праћењем поремећаја у конзистентности података.

Слика 26: Студентска служба

Користећи ССА модел процеса и посебно дијаграме тока података, као и познавање реализованог развоја решења, као и елементе дизајна - логичког дизајна и имплементације - физичког дизајна решења, за све идентификоване грешке, анализирани су појединачни узроци и одређена је припадност узрока одређеној области информационог система: хардвер, софтвер, мрежа или процедуре. Узроци идентификованих грешака, и не само грешака обухваћених овим истраживањем, посебно ако су узроковане пропустима у фази дизајна или због непоштовања најбољих пракси - препоручених пројектних образаца, уведени су у интерну базу података и категорисани, како би служили као део организацијског знања о томе. Приликом анализе потенцијалних ефеката идентификованих грешака, констатовано је да ефекти-последице могу бити локалног нивоа и на нивоу подсистема и на нивоу система. Овај аспект посебно је утицао на процену озбиљности сваке идентификоване грешке.

5.3.6.2. Оцена важности фактора ризика

За сваку идентификовану грешку процењивана су три фактора ризика (РФ-а): озбиљност последице која настаје услед реализације грешке (С) - Severity, вероватноћа настајања грешке (О) - Occurrence, и могућност откривања грешке (Д) - Detection. Најпре је било потребно одредити међусобни однос важности ових ризика, као што је то предвиђено методологијом. Ово је значајна разлика у односу на класичну FMEA.

Релативна важност ових фактора, као и касније саме вредности фактора ризика за све идентификоване грешке, одређене су на основу података из званичних евиденција и стечених искустава при изради и одржавању овог и сличних система.

Треба истаћи да је процена важности фактора ризика обављена стриктно узимајући у обзир конкретне идентификоване грешке. Претпоставка је да би за други скуп идентификованих грешака та процена могла бити другачија. Као што је предвиђено алгоритмом предложене методологије, најпре су идентификоване грешке посматраног софтвера, па је тек потом спроведена оцена важности РФ-а.

Релативна важност РФ-а се по методологији задаје матрично. У овом истраживању међусобно поређење важности РФ-а описано је помоћу три лингвистичка термина који су моделирани IT2TrFN:

- *мала важност (Л)*-$\big((1,1,2,5;1),(1,1,2,4;0.75)\big)$
- *средња важност (М)*-$\big((1.5,2.5,3.5,4.5;1),(2,2.5,3.5,4;0.75)\big)$
- *висока важност (Н)*-$\big((1,4,5,5;1),(2,4,5,5;0.75)\big)$

Применом ових лингвистичких исказа, формирана је следећа матрица парног поређења важности фактора ризика за идентификоване грешке:

$$\begin{bmatrix} \tilde{\tilde{1}} & MW & HW \\ & \tilde{\tilde{1}} & LW \\ & & \tilde{\tilde{1}} \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 1.25 & 3.708 \\ & 1 & 2.625 \\ & & 1 \end{bmatrix},\ C.I. = 0.0025$$

Коефицијент конзистентности матрице поређења показао је да је усклађеност процена у границама које су дозвољене.

Израчунавање фази вектора тежине РФ-а, предвиђено је 3. кораком у алгоритму предложене методологије, а у складу са [52].

$$\tilde{\tilde{r}}_1 = \begin{pmatrix} (\sqrt[3]{1 \cdot 1.5 \cdot 1}, \sqrt[3]{1 \cdot 2.5 \cdot 4}, \sqrt[3]{1 \cdot 3.5 \cdot 5}, \sqrt[3]{1 \cdot 4.5 \cdot 5};\ 1{,}1) \\ , (\sqrt[3]{1 \cdot 2 \cdot 2}, \sqrt[3]{1 \cdot 2.5 \cdot 4}, \sqrt[3]{1 \cdot 3.5 \cdot 4}, \sqrt[3]{1 \cdot 4 \cdot 5};\ 0.75{,}0.75) \end{pmatrix} =$$
$$((1.143, 2.138, 2.571, 2.794;\ 1{,}1), (1.580, 2.138, 2.571, 2{,}687;\ 0.75, 0.75))$$

На сличан начин рачунамо даље:

$$\tilde{\tilde{r}}_2 = ((0.588, 0.795,\ 1.257, 1.701;\ 1{,}1), (0.637,\ 0.795, 1.257, 1.584;\ 0.75, 0.75))$$

$$\tilde{\tilde{r}}_3 = ((0.346, 0.468,\ 0.633, 1;\ 1{,}1), (0.372,\ 0.468, 0.633, 0.796;\ 0.75, 0.75))$$

$$\tilde{\tilde{R}} = \tilde{\tilde{r}}_1 + \tilde{\tilde{r}}_2 + \tilde{\tilde{r}}_3 = \begin{pmatrix} (2.077, 3.401, 4.461, 5.495;\ 1{,}1) \\ , (2{,}584, 3.401, 4.461, 5.063;\ 0.75, 0.75) \end{pmatrix}$$

Тежински вектор фактора ризика рачунамо на следећи начин:

$$\tilde{\tilde{w}}_1 = \begin{pmatrix} (1.143/5.495, 2.138/4.461, 2.571/3.401, 2.794/2.077;\ 1{,}1), \\ (1.580/5.063, 2.138/4.461, 2.571/3.401, 2{,}687/2.584;\ 0.75, 0.75) \end{pmatrix}$$

$$\tilde{\tilde{w}}_1 = ((0.21, 0.48, 0.76, 1.34;\ 1{,}1), (0.31, 0.48, 0.76, 1.04;\ 0.75, 0.75))$$

Тежинске векторе остала два фактора ризика рачунамо на исти начин:

$$\tilde{\tilde{w}}_2 = ((0.11, 0.18, 0.37, 0.82;\ 1{,}1), (0.12, 0.18, 0.37, 0.61;\ 0.75, 0.75))$$

$$\tilde{\tilde{w}}_3 = ((0.06,\ 0.10,\ 0.19,\ 0.48;\ 1{,}1), (0.07, 0.10, 0.19, 0.31;\ 0.75, 0.75))$$

5.3.6.3. Одређивање вредности РФ-а идентификованих грешака

Доносиоци одлука, у овом случају који су учествовали у развоју и одржавању посматраног софтвера, проценили су вредности РФ-а идентификованих грешака. Процене су засноване на карактеристикама овог и сличних софтверских решења, историјским подацима из базе података, дневника одржавања, подацима из рада софтверских контрола, проценама од стране крајњих корисника система и слично.

Озбиљност последица је субјективна оцена. На посматраном примеру софтвера студентске службе факултета, од посебног значаја за оцену озбиљности последица, било је питање на колико велики део система одређена грешка утиче. Грешке за које је утврђено да имају последице и ван процеса у којем су настале, а посебно грешке које утичу на цео систем, добиле су по правилу високе оцене озбиљности. Озбиљност сваке последице која настаје услед материјализације сваке грешке се у конвенционалној FMEA методи оцењује само са аспекта квалитета. Предложеном методологијом у овом докторском раду предвиђено је да се последице које настају услед реализације грешака софтвера оцењују са аспекта протока података у процесима ИС-а.

Коришћени лингвистички искази за озбиљност последица:

- *Без последица на проток података (СД1)*
- *Врло мали утицај на проток података (СД2)*
- *Мали утицај на проток података (СД3)*
- *Средњи утицај на проток података (СД4)*
- *Велики утицај на проток података (СД5)*
- *Врло велики утицај на проток података (СД6)*
- *Екстремно велики утицај на проток података (СД7)*

Учесталост настајања идентификованих грешака одређена је на основу процене доносилаца одлука. За део идентификованих грешака при одређивању процене учесталости било је могуће на довољно добар начин користити податке из евиденција из фазе одржавања софтвера. Обезбеђивање прецизних бројева о учесталости настајања већине идентификованих грешака у будућем периоду решено је креирањем и имплементацијом одређеног броја софтверских контрола.
Коришћени лингвистички искази за учесталост настајање грешака:

- *Веома ретко (О1)*
- *Ретко (О2)*
- *Периодично (О3)*
- *Често (О4)*
- *Веома често (О5)*

Могућност детекције грешака је оцењена на основу анализе да ли за посматране грешке у оквиру информационог система постоје развијене процедуре детекције и да ли је у софтверу процедура имплементирана. Вредност додељена овом фактору ризика је такође субјективна оцена. Неке методе детекције које су идентификоване у посматраном систему су и плански преглед сумарних података у бази података, инспекција прекинутих сложених обрада, пријаве крајњих корисника. Узете су у обзир све идентификоване методе детекције ради оцене могућности детекције.

Коришћени лингвистички искази за могућност детекције грешака:

- *Није могуће детктовати (СД1)*
- *Готово немогуће детектовати(СД2)*
- *Врло мала могућност детекције (СД3)*
- *Средње мала могућност детекције (СД4)*
- *Средње велика могућност детекције (СД5)*
- *Велика могућност детекције (СД6)*
- *Апсолутна ефикасност детекције (СД7)*

Процена фактора ризика за идентификоване грешке дата је у табели 13.

Табела 13: Оцене РФ-а идентификованих грешака

Р.б.	*Озбиљност*	*Учесталост*	*Детекција*
1.	*СД4*	*О3 (10)*	*СД5*
2.	*СД3*	*О3 (15)*	*СД4*
3.	*СД4*	*О3 (12)*	*СД5*
4.	*СД3*	*О4 (22)*	*СД4*
5.	*СД2*	*О2 (5)*	*СД4*
6.	*СД2*	*О2*	*СД3*
7.	*СД5*	*О2 (5)*	*СД4*
8.	*СД4*	*О3 (30)*	*СД3*
9.	*СД2*	*О1 (2)*	*СД3*
10.	*СД3*	*О4*	*СД2*
11.	*СД5*	*О4 (50)*	*СД6*
12.	*СД6*	*О4 (129)*	*СД6*
13.	*СД3*	*О4*	*СД7*
14.	*СД2*	*О3*	*СД7*
15.	*СД1*	*О3 (14)*	*СД6*
16.	*СД2*	*О5*	*СД7*
17.	*СД4*	*О3 (30)*	*СД4*
18.	*СД2*	*О3*	*СД6*
19.	*СД4*	*О3*	*СД2*
20.	*СД2*	*О2*	*СД4*
21.	*СД3*	*О2 (10)*	*СД5*
22.	*СД2*	*О2 (13)*	*СД3*
23.	*СД2*	*О1 (1)*	*СД4*
24.	*СД3*	*О2 (20)*	*СД3*
25.	*СД2*	*О2 (10)*	*СД6*
26.	*СД3*	*О1 (1)*	*СД6*
27.	*СД5*	*О2 (2)*	*СД6*

28.	*СД4*	*О2 (15)*	*СД6*
29.	*СД3*	*О3 (47)*	*СД6*
30.	*СД3*	*О3 (46)*	*СД6*
31.	*СД5*	*О2 (7)*	*СД6*
32.	*СД5*	*О2 (11)*	*СД7*
33.	*СД5*	*О1 (4)*	*СД2*
34.	*СД4*	*О2 (30)*	*СД6*
35.	*СД3*	*О4*	*СД3*
36.	*СД3*	*О3*	*СД3*
37.	*СД3*	*О3 (40)*	*СД3*
38.	*СД3*	*О2 (5)*	*СД3*
39.	*СД3*	*О3 (50)*	*СД5*
40.	*СД1*	*О2*	*СД3*
41.	*СД2*	*О4 (100)*	*СД5*

5.3.6.4. Израчунавање ранга грешака

Ранг идентификованих грешака софтвера је одређен применом конвенционалне COPRAS методе [53].

Најпре је било потребно применити раније израчунате векторе тежина РФ-а и формирати отежану матрица одлучивања.

$$\left[\tilde{\tilde{d}}_{jk}\right]_{JxK}, j = 1,..,J; k = 1,\ldots,K.$$

Појединачни чланови матрице израчунати су по процедури из предложене методологије, као у овом примеру:

$$\tilde{\tilde{d}}_{11} = \tilde{\tilde{w}}_1 \cdot \tilde{\tilde{v}}_{11} = \begin{pmatrix} (0.21 \cdot 0.3, 0.48 \cdot 0.4, 0.76 \cdot 0.5, 1.34 \cdot 0.6;\ 1,1), \\ ((0.31 \cdot 0.35, 0.48 \cdot 0.4, 0.76 \cdot 0.5, 1.04 \cdot 0.55;\ 0.75, 0.75)) \end{pmatrix}$$

$$\tilde{\tilde{d}}_{11} = \begin{pmatrix} (0.063, 0.192, 0.380, 0.804;\ 1,1) \\ , ((0.108, 0.192, 0.380, 0.572;\ 0.75, 0.75)) \end{pmatrix}$$

Да би се спровела конвенционална COPRAS метода примењен је поступак дефазификације предложен у [52].

Сви појединачни репрезентативни скалари од IT2TrFN, $\tilde{\tilde{d}}_{ik}$ су израчунати као у следећем примеру:

$$d_{11}$$

$$= \frac{1}{2} \cdot \left\{ \begin{array}{l} \dfrac{(0.804 - 0.063) + (1 \cdot 0.192 - 0.063) + (1 \cdot 0.380 - 0.063)}{4} + 0.063 + \\ \dfrac{(0.572 - 0.108) + (0.75 \cdot 0.192 - 0.108) + (0.75 \cdot 0.380 - 0.108)}{4} + 0.108 \end{array} \right\}$$

$$= 0.325$$

На тај начин добили смо дефазификовану матрицу одлучивања која је приказана у табели 14.

Табела 14: Матрица одлучивања

	С	О	Д		С	О	Д
i=1	0.325	0.195	0.074	i=22	0.195	0.129	0.128
i=2	0.260	0.195	0.092	i=23	0.195	0.053	0.092
i=3	0.325	0.195	0.074	i=24	0.260	0.129	0.128
i=4	0.260	0.293	0.092	i=25	0.195	0.129	0.056
i=5	0.195	0.129	0.092	i=26	0.260	0.053	0.056
i=6	0.195	0.129	0.128	i=27	0.454	0.129	0.056
i=7	0.454	0.129	0.092	i=28	0.325	0.129	0.056
i=8	0.325	0.195	0.128	i=29	0.260	0.195	0.056
i=9	0.195	0.053	0.128	i=30	0.26	0.195	0.056
i=10	0.26	0.293	0.146	i=31	0.454	0.129	0.056
i=11	0.454	0.293	0.056	i=32	0.454	0.129	0.036
i=12	0.519	0.293	0.056	i=33	0.454	0.053	0.146
i=13	0.260	0.293	0.036	i=34	0.325	0.129	0.056
i=14	0.195	0.195	0.036	i=35	0.260	0.293	0.128
i=15	0.116	0.195	0.056	i=36	0,26	0.195	0.128
i=16	0.195	0.315	0.036	i=37	0.260	0.195	0.128
i=17	0.325	0.195	0.092	i=38	0.260	0.129	0.128
i=18	0.195	0.195	0.056	i=39	0.260	0.195	0.074
i=19	0.325	0.195	0.146	i=40	0.116	0.129	0.128
i=20	0.195	0,129	0,092	i=41	0.195	0.293	0.074
i=21	0.260	0.129	0.074				

На нивоу сваке идентификоване грешке одређене су агрегиране вредности бенефитних РФ-а, P_i и трошковних РФ-а, Q_i, као и укупне агрегиране вредности Z_i, као у следећем примеру:

$$Z_1 = 0.074 + \frac{0.245 \cdot 18.778}{0.520 \cdot 23.661} = 0.448$$

Добијене вредности приказане су у табели 15.

Табела 15: Агрегиране вредности и укупне агрегиране вредности

	P_j	Q_j	Z_j		P_j	Q_j	Z_j
i=1	0.074	0.520	0.448	i=22	0.128	0.324	0.728
i=2	0.092	0.455	0.519	i=23	0.092	0.248	0.876
i=3	0.074	0.520	0.448	i=24	0.128	0.389	0.628
i=4	0.092	0.553	0.444	i=25	0.056	0.324	0.656
i=5	0.092	0.324	0.692	i=26	0.056	0.313	0.677
i=6	0.128	0.324	0.728	i=27	0.056	0.583	0.389
i=7	0.092	0.583	0.426	i=28	0.056	0.454	0.484
i=8	0.128	0.520	0.502	i=29	0.056	0.455	0.483
i=9	0.128	0.248	0.912	i=30	0.056	0.455	0.483
i=10	0.146	0.553	0.498	i=31	0.056	0.583	0.389
i=11	0.056	0.747	0.316	i=32	0.036	0.583	0.374
i=12	0.056	0.812	0.295	i=33	0.146	0.507	0.529
i=13	0.036	0.553	0.388	i=34	0.056	0.454	0.484
i=14	0.036	0.390	0.535	i=35	0.128	0.553	0.480
i=15	0.056	0.311	0.681	i=36	0.128	0.455	0.555
i=16	0.036	0.510	0.417	i=37	0.128	0.455	0.556
i=17	0.092	0.520	0.466	i=38	0.128	0.389	0.628

i=18	0.056	0.390	0.555	i=39	0.074	0.455	0.501
i=19	0.146	0.520	0.520	i=40	0.128	0.245	0.922
i=20	0.092	0.324	0.692	i=41	0.074	0.488	0.472
i=21	0.074	0.389	0.574				

Мера корисности за сваку идентификовани грешку i, i=1,...,41 и ранг сваке грешке је израчунат према предложеној методологији као на примеру:

$$\zeta_1 = \frac{0.448}{0.922} = 0.486$$

Табела 16 приказује ове вредности за све идентификоване грешке и представља резултат примењене COPRAS методе. Ранг свих идентификованих грешака на нивоу сваког примитивног процеса је одређен респектујући израчунате вредности ζ_j. На првом месту у рангу, односно на последње месту у рангу, респективно налази се она грешка којој је придружена највећа, односно најмања вредност ζ_j, респективно.

Табела 16: Мера корисности и ранг грешака

	ζ_i	Rang		ζ_i	Rang		ζ_i	Rang
i=1	0.486	10-11	i=15	0.739	34	i=29	0.524	15-18
i=2	0.563	22	i=16	0.452	7	i=30	0.524	15
i=3	0.486	10-11	i=17	0.505	12	i=31	0.422	5-6
i=4	0.482	9	i=18	0.602	26	i=32	0.406	3
i=5	0.751	35-36	i=19	0.564	23	i=33	0.574	25
i=6	0.789	37-38	i=20	0.751	35-36	i=34	0.525	15-18
i=7	0.462	8	i=21	0.623	29	i=35	0.521	14
i=8	0.544	21	i=22	0.790	37-38	i=36	0.602	27-28
i=9	0.989	40	i=23	0.950	39	i=37	0.603	27-28
i=10	0.540	19-20	i=24	0.681	30-31	i=38	0.681	30-31
i=11	0.346	2	i=25	0.711	32	i=39	0.543	19-20

i=12	0.320	1	i=26	0.734	33	i=40	1	41
i=13	0.421	4	i=27	0.422	5-6	i=41	0.512	13
i=14	0.580	24	i=28	0.525	15-18			

5.3.7. Одређивање и примена мера за отклањање грешака

При одређивању мера у највећој мери је коришћено познавање развијеног софтвера и информационог система, као и модели развијени у оквиру спроведене ССА, што је битан елемент предложене методологије у овој дисертацији. Дефинисане су мере које треба да смање ниво ризика који настаје услед идентификованих грешака из FMEA анализе. Користећи искуство из претходно спроведених итерација развоја посматраног софтвера дефинисана су побољшања и исправке актуелне верзије софтверског решења као мере које треба да доведу до смањења ризика и потенцијално до повећања протока података у посматраном информационом систему.

Смањивање или елиминисање утицаја идентификованих грешака на функционисање софтвера може да се постигне предузимањем одговарајућих мера. У овом истраживању, редослед примене мера базиран је на установљеном рангу идентификованих грепака. Примена овог приступа, омогућава доносиоцима одлуке да потроше мање ресурса (новца, времена,..) а истовремено да повећају ефикасност софтвера.

Како се од четири анализиране грешке које имају највећи утицај на функционисање разматраног софтвера, три грешке (и=11, и=12, и=13) материјализују у процесу који је означен као "*Прикупљање и анализа жеља студената"*, развојни тим треба посебно да обрати пажњу на део информационог система који се односи на овај процесс. То даље значи да је неопходно да се провере све фазе развоја овог процеса.

У овом истраживању, а на основу идентификованих узрока грешака из спроведене FMEA, препознате су две врсте мера које се могу предузети према нивоу припадности посматраном систему. Прва и основна врста су мере на модификацијама, дорадама и исправкама софтвера, како на нивоу логичког и физичког дизајна, тако и на нивоу имплементације - примењених технологија, хардвера и слично. Друго су мере на дефинисању и изменама процедура и реалних процеса који утичу на процесе информационог система. Када су у питању

мере које отклањају пропусте у софтверу, дефинисане су мере које спречавају настанак услова за испољавање грешака, мере које контролишу и спречавају сам догађај грешке и мере које ублажавају последице грешака. У случајевима где грешке настају због фактора ван софтвера уже посматрано, због грешака или непоштовања процедура од стране корисника, или због грешака из окружења, тежиште је било на мерама које умањују последице.

За неке грешке су дефинисане вишеструке мере. Однос појединачних идентификованих грешака и мера за отклањање грешака је у релацији више-према-више. За сваку идентификовану грешку, одређена је минимално једна мера.

Мере су одређене и са фазом софтверског процеса у којем треба да буду примењене. Није свеједно да ли мера треба да се примени у фази логичког дизајна, да ли је у питању грешка приликом моделовања система или је потребна нека минорна поправка кода. Мера дефинисана за примену у ранијим фазама софтверског процеса подразумева потребне интервенције и у каснијим фазама. На пример свака промена у моделу података сасвим сигурно захтева и модификације у физичкој имплементацији модела у конкретном систему базе података. Оваква модификација и дорада софтвера може се посматрати као још једна итерација развоја. Развојни тим је имајући у виду да имамо фазе анализе, дизајна, имплементације и експлоатације у коришћеном моделу софтверског процеса, тражио решење проблема у што каснијим фазама модела. У табели 17, приказане су дефинисане мере за отклањање грешака.

Приликом дефинисања мера, поставило се и питање домета мера. Мере су дефинисане уже, пре свега са фокусом на конкретну грешку коју мера треба да реши, не широко са потенцијално већим утицајем на систем али и са већим захтевима у потребним ресурсима за реализацију.

Табела17: Мере за отклањање грешака

Грешка	**Процес**	**Мера**	**Фаза**	**сати рада**
1.	3.1.2.1.	Редефинисати односе између класа корисничког интерфејса. Унапредити класу за табеларни приказ. Применити шаблон Dependency injection.	Имплементација	16

2.	3.1.2.1.	Додавање додатне софтверске контроле у одговорном блоку кода.	Кодирање	4
3.	3.1.2.1.	Додељивање атрибута за истицање у коду погледа одговорајућег MVC шаблона.	Кодирање	2
4.	3.1.2.2.	Додавање ограничења у одговарајућој табели у бази података.	Имплементација	1
5.	3.1.2.2.	Допуна модела података редним бројем записа у табели извођача. Прилагођавање кода за упис податка.	Дизајн	4
6.	3.1.3.1.	Креирање нове класе за управљање модулима. Додавање одговарајућих опција у кориснички интерфејс и додељивање права корисницима.	Имплементација	24
7.	3.1.3.2.	Додавање додатне софтверске контроле у одговорном блоку кода.	Кодирање	8
8.	3.1.3.3.	Исправка SQL команде за фотмирање скупа података за експорт ка студентском сервису.	Кодирање	1
9.	3.1.3.3.	Креирање нове класе за управљање модулима. Додавање одговарајућих опција у кориснички интерфејс и додељивање права корисницима.	Имплементација	24
10.	3.1.4.1.	Додавање новог извештаја модулу за извештавање. Додавање одговарајућих опција у кориснички интерфејс и додељивање права корисницима.	Имплементација	16
11.	3.1.4.1.	Промена процедуре провере испињености услова за доделу предмета студента. Модификација логике рада методе за обраду студентских жеље.	Имплементација	24
12.	3.1.4.1.	Додавање потребне контроле у кориснички интерфејс.	Кодирање	8
13.	3.1.4.1.	Проширење модела података подсистема задуженог за интерфејс. Модификовати методу одговарајуће класе за експорт података у базу података подсистема.	Имплементација	8
14.	3.1.4.1.	Повећати фреквенцију синхронизације база података укључених у процес.	Одржавање	8
15.	3.1.4.2.	Рефакторисање кода за обраду захтева. Пребацити обраду података из релационе базе у радну меморију, користити већ креиране ORM класе.	Кодирање	40

16.	3.1.4.3.	Модификовати руковалац класу да обезбеди функционалност понављања доделе предмета. Подесити правилно контролер елемент MVC шаблона.	Кодирање	16
17.	3.1.4.3.	Модификовати класе корисничког интерфејса да обезбеди приказ. Користити observer шаблон пројектовања.	Имплементација	24
18.	3.1.4.4.	Издвојити класу за извештавање у самосталну библиотеку. Уградити потребну функционалност у основну апликацију.	Имплементација	16
19.	3.2.1.1.	Промена процедуре евидентирања потписа. Модификација методе одговарајуће класе са респектовањем активне школске године.	Кодирање	16
20.	3.2.1.1.	Промена процедуре евидентирања потписа. Модификација методе одговарајуће класе са респектовањем активне школске године и године увођења предмета.	Кодирање	16
21.	3.2.1.2.	Уклонити контролу грешке из кода методе одговарајуће класе. Кодирати исправно понашање респектујући школску годину.	Кодирање	8
22.	3.2.1.3.	Креирати нови комбиновани извештај. Дефинисати процедуру за коришћење информација из новог извештаја.	Имплементација	32
23.	3.2.2.2.	Модификовати логику рада методе одговрајуће класе. Увести софтверску контролу параметара корисничког захтева.	Кодирање	16
24.	3.2.2.3.	Креирати табелу еквиваленције за исте предмете са различитим идентификатором. Модификовати класу за управљање испитима да респектује нову табелу.	Имплементација	40
25.	3.2.2.3.	Увести софтверску контролу броја записа пријаве испита. Модификовати методу класе за генерисање интерног XML формата.	Кодирање	24
26.	3.2.2.3.	Додати у модел података ентитет блока испитног рока. Модификовати логику обрачуна броја пријава у класи за управљање полагањем испита.	Дизајн	24
27.	3.2.2.4.	Увести одговарајуће ограничење табеле са списком испита у бази података. Унети подразумеване вредности у раније записе.	Имплементација	4

28.	3.2.2.4.	Увести софтверску контролу броја записа пријаве испита. Модификовати методу класе за генерисање интерног XML формата.	Кодирање	24
29.	3.2.2.5.	Увести контролну маску у елемент корисничког интерфејса за контролу формата податка.	Кодирање	4
30.	3.2.2.5.	Проширити позив методе класе за директан унос оцена са параметром валидности предмета. Применити dekorator шаблон.	Кодирање	8
31.	3.2.2.5.	Додати опциону функционалност криптовања модулу за формирање XMF фајлова.	Кодирање	24
32.	3.2.2.5.	Увести контролу компатибилности верзије алата за offline унос оцена и верзије фајла са оценама.	Кодирање	8
33.	3.2.2.5.	Додати опциону функционалност криптовања модулу за формирање XMF фајлова.	Кодирање	24
34.	3.2.2.5.	Проширити позив методе класе за директан унос оцена са параметром валидности статуса студента. Применити dekorator шаблон.	Кодирање	8
35.	3.2.2.6.	Креирати нови SQL упит за генерисање исправне анализе. Обезбедити проверу агрегираних вредности.	Кодирање	16
36.	3.2.3.2.	Увести фази софтверску контролу за проверу броја ЕСПБ-а студента према различитим параметрима.	Имплементација	16
37.	3.2.3.2.	Увести софтверску контролу компатибилности положенох испита и студијских програма и модула. Спровести миграцију некомпатибилних историјских података у издвојене структуре података.	Имплементација	24
38.	3.2.3.3.	Потврдити процедуру за израчунавање просечне оцене студента. Променити логику методе за унос података класе за управљање дипломирањем.	Имплементација	32
39.	3.2.3.3.	Проширити модел података са опционим додатним податком датума испита. Увести софтверску контролу провере редоследа положених испита.	Имплементација	24
40.	3.2.3.3.	Проширити модел података са обавезним податком редног броја члана комисије.	Кодирање	8
41.	3.2.3.3.	Увести софтверску контролу провере редоследа положених испита.	Имплементација	8

Дефинисане мере су реализоване у више фаза имплементације. Свака фаза је резултирала новом верзијом софтвера. Мере су обухватиле разноврсне модификације и унапређења информационог система и пре свега софтвера. Већи део предвиђених мера уграђен је у верзију софтвера која је уведена у употребу 1.5.2018. године. Остатак предвиђених мера је имплементиран у следећој верзији која је уведена у употребу 1.6.2018. године.

Мере које су позициониране у фазу дизајна и кодирања, узимајући у обзир да је при изради софтвера доминантно коришћен објектно оријентисан програмски језик C#, често у свом опису садрже референцу на неки пројектни шаблон, образац (design pattern). Пројектни шаблон је вишекратно употребљиво решење за познат проблем. Шаблони су начин да се поново употреби знање и искуство из других пројеката. Пројектант који познаје пројектне шаблоне има предност при решавању проблема, нема потребе да поново открива најбоља решења, уместо тога примењује решења која су већ доказана.

Пројектни шаблони представљају примере решења проблема на високом нивоу апстракције. Они омогућавају изражавање намера кроз заједнички речник. Уколико чланови развојног тима познају одређене пројектне шаблоне, то им омогућава ефикасну и квалитетну комуникацију без непотребног оптерећивања имплементационим појединостима.

Осим боље комуникације развојног тима, пројектни шаблони, ако се правилно користе, олакшавају измене система. Обезбеђују да систем може да се на једноставнији, сигурнији и јефтинији начин мења. Циљ је да се промене неког елемента система могу спровести независно од других елемената.

Приликом пројектовања и програмирања посматраног софтвера, коришћени су разни пројектни шаблони, већина је из традиционалног скупа GoF пројектних шаблона [89].

Редослед реализације мера одређен је најпре узимајући у обзир ранг грешака, а потом и фазу у софтверском процесу. Редослед реализације мера, а који је примарно зависио од ранга грешака као резултата предходне фазе методологије, показао се као веома важан због процеса поступног спровођења и

редефинисања мера и потребних модификација софтвера респектујући последице раније примењених мера. Редослед реализације мера имао је значајан утицај на цео процес.

Није примењен опциони поступак проналажења оптималног подскупа мера на осову процене од стране развојног тима да је скуп мера довољно мали и да се може комплетно реализовати у предвиђеном времену и ресурсима, што се и показало као тачно. Тражити оптимални подскуп мера које ће се реализовати би вероватно било потребно да су експериментом обухваћени сви процеси посматраног система. У том случају би се морала спровести и процена потенцијалних и очекиваних ефеката мера, као и процена утрошка ресурса за реализацију мера, како би се могао одредити оптималан подскуп мера.

5.3.8. Оцена ефеката предузетих мера

Вредности дефинисаних индикатора за период пре и после примене корективних мера, у периоду коришћења претходне и модификоване верзије, приказане су у табели 18.

Вредности индикатора су измерене за периоде истог временског трајања и са поклапањем циклуса активности и одвијања реалних процеса на факултету. Спецификација појединачних индикатора остала је иста за оба посматрана периода. Техника мерења вредности за један број индикатора је унапређена у току целог процеса али то није утицало на установљене вредности индикатора.

Посматрано на нивоу процеса, са све процесе респектујући вредности индикатора, остварен је одређен напредак.

Табела18:Упоредне вредности индикатора за контролни период и период после примене мера и увођења нове верзије софтвера у употребу

Процес	**Индикатор**	**Вредност1**	**Вредност2**	**Смер поз.пром.**	**Разлика**	**Промена**
3.1.1.	1	5	5	1	0	0,00%
3.1.2.1.	1	1500	1670	1	170	11,33%
3.1.2.1.	2	72	88	1	16	22,22%
3.1.2.1.	3	21	20	-1	1	4,76%
3.1.2.2.	1	4952	5213	1	261	5,27%
3.1.2.2.	2	390	392	1	2	0,51%
3.1.2.2.	3	45	19	-1	26	57,78%
3.1.2.3.	1	185	192	1	7	3,78%
3.1.3.1.	1	24	26	1	2	8,33%
3.1.3.2.	1	750	798	1	48	6,40%
3.1.3.3.	1	2254	2370	1	116	5,15%

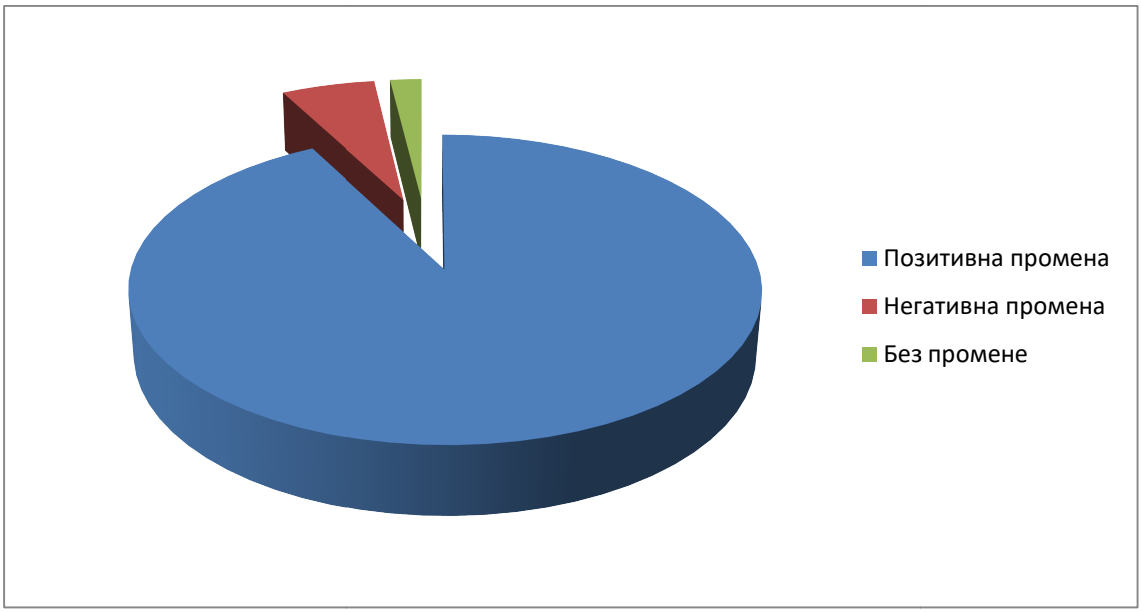
3.1.4.1.	1	26080	28250	1	2170	8,32%
3.1.4.1.	2	1860	1800	-1	60	3,23%
3.1.4.2.	1	19990	22640	1	2650	13,26%
3.1.4.2.	2	12000	13200	1	1200	10,00%
3.1.4.2.	3	6000	5980	1	-20	-0,33%
3.1.4.2.	4	1990	1850	1	-140	-7,04%
3.1.4.2.	5	0,623	0,654	1	0,031	4,98%
3.1.4.3.	1	2650	2750	1	100	3,77%
3.1.4.4.	1	2200	2780	1	580	26,36%
3.1.4.4.	2	5000	6200	1	1200	24,00%
3.1.4.4.	3	1,24	0,89	-1	0,35	28,23%
3.2.1.1.	1	17500	18900	1	1400	8,00%
3.2.1.1.	2	17150	18888	1	1738	10,13%
3.2.1.2.	1	50	60	1	10	20,00%
3.2.1.3.	1	1500	1550	1	50	3,33%
3.2.1.3.	2	1,33	1,12	-1	0,21	15,79%
3.2.1.3.	3	2,5	2,49	-1	0,01	0,40%
3.2.2.1.	1	30	32	1	2	6,67%
3.2.2.1.	2	145,5	120	-1	25,5	17,53%
3.2.2.1.	3	2,2	2,18	-1	0,02	0,91%
3.2.2.2.	1	25000	27000	1	2000	8,00%
3.2.2.2.	2	20000	21000	1	1000	5,00%
3.2.2.2.	3	0,02	0,02	-1	0	0,00%
3.2.2.2.	4	3,8	3,79	-1	0,01	0,26%
3.2.2.3.	1	295000	310000	1	15000	5,08%
3.2.2.3.	2	180000	195000	1	15000	8,33%
3.2.2.3.	3	1,56	1,45	-1	0,11	7,05%

3.2.2.3.	4	3,7	3,2	-1	0,5	13,51%
3.2.2.4.	1	24990	26850	1	1860	7,44%
3.2.2.4.	2	289000	305000	1	16000	5,54%
3.2.2.4.	3	13000	13040	-1	-40	-0,31%
3.2.2.5.	1	24985	26770	1	1785	7,14%
3.2.2.5.	2	288842	304020	1	15178	5,25%
3.2.2.5.	3	102555	106000	1	3445	3,36%
3.2.2.5.	4	275	199	-1	76	27,64%
3.2.2.6.	1	6850	7800	1	950	13,87%
3.2.2.6.	2	1,1	1,09	-1	0,01	0,91%
3.2.3.1.	1	750	770	1	20	2,67%
3.2.3.2.	1	45,5	39,5	-1	6	13,19%
3.2.3.3.	1	742	866	1	124	16,71%
3.2.3.3.	2	0,085	0,064	-1	0,021	24,71%

За 52 посматрана индикатора, анализа резултата показала је

- У 48 случајева дошло је до позитивне промене респектујући позитиван смер промене вредности индикатора,
- У односу на прво мерење, у другом је дошло до промене вредности 48 индикатора која нам указује на повећање учинка процеса или повећање учинка по активном времену процеса са аспекта протока података.
- У једном случају је вредност индикатора остала непромењена.
- У само 3 од 52 случаја је констатована негативна промена вредности индикатора. У два случаја је степен негативне промене занемарљив, а за један индикатор где је степен негативне промене 7,04%.

Графичка представа удела броја индикатора са врстом промене у укупном скупу је приказана на слици 27.



Слика 27:Удео врста промене вредности индикатора пре и после

Анализа података на основу којих је обрачунат овај индикатор показала је да је разлог промене последица догађаја вам информационог система и такође указала је на потребу да се редефинише индикатор како би боље указивао на успешност процеса а не на једноставан број обрада. Проценат апсолутно позитивних промена индикатора на овом скупу података је 92,31%. Ако се узме као адекватан закључак да је само један индикатор показао значајнију промену на горе, то је 1,92% погоршаних индикатора примитивних процеса. За конкретан процес дефинисано је 5 индикатора од којих је за три измерена позитивна промена индикатора.

Упоредни приказ иницијалне процене фактора ризика и процене после једногодишњег периода рада система са унапређеним софтвером на основу идентификованих и примењених мера побољшања система, је приказан у табели 19. Процене су засноване на карактеристикама овог софтвера, историјским подацима из базе података, дневника одржавања, подацима из рада софтверских контрола и мишљења крајњих корисника система.

Табела 19:Упоредне оцене РФ-а пре и после примене мера и увођења нове верзије софтвера у употребу

Грешка	**Озбиљност**	**Учесталост**	**Детекција**	**Озбиљност после прим. мера**	**Учесталост после прим. мера**	**Детекција после прим.мера**
1.	СД4	О3	СД5	СД3	О1	СД6
2.	СД3	О3	СД4	СД2	О2	СД6
3.	СД4	О3	СД5	СД2	О2	СД5
4.	СД3	О4	СД4	СД1	О2	СД7
5.	СД2	О2	СД4	СД2	О1	СД5
6.	СД2	О2	СД3	СД1	О1	СД5
7.	СД5	О2	СД4	СД2	О2	СД4
8.	СД4	О3	СД3	СД2	О1	СД5
9.	СД2	О1	СД3	СД1	О1	СД7
10.	СД3	О4	СД2	СД3	О2	СД7
11.	СД5	О4	СД6	СД2	О3	СД6
12.	СД6	О4	СД6	СД2	О3	СД7
13.	СД3	О4	СД7	СД2	О2	СД7
14.	СД2	О3	СД7	СД1	О1	СД7
15.	СД1	О3	СД6	СД1	О2	СД6
16.	СД2	О5	СД7	СД1	О3	СД7
17.	СД4	О3	СД4	СД2	О2	СД7
18.	СД2	О3	СД6	СД2	О2	СД6
19.	СД4	О3	СД2	СД2	О1	СД7
20.	СД2	О2	СД4	СД1	О2	СД7
21.	СД3	О2	СД5	СД1	О1	СД7
22.	СД2	О2	СД3	СД2	О2	СД7
23.	СД2	О1	СД4	СД1	О1	СД7
24.	СД3	О2	СД3	СД3	О1	СД6
25.	СД2	О2	СД6	СД2	О1	СД6
26.	СД3	О1	СД6	СД2	О1	СД6

27.	СД5	О2	СД6	СД2	О2	СД6
28.	СД4	О2	СД6	СД1	О2	СД6
29.	СД3	О3	СД6	СД2	О2	СД6
30.	СД3	О3	СД6	СД1	О1	СД7
31.	СД5	О2	СД6	СД2	О2	СД6
32.	СД5	О2	СД7	СД1	О1	СД7
33.	СД5	О1	СД2	СД1	О1	СД5
34.	СД4	О2	СД6	СД2	О2	СД6
35.	СД3	О4	СД3	СД3	О3	СД7
36.	СД3	О3	СД3	СД2	О2	СД7
37.	СД3	О3	СД3	СД1	О2	СД7
38.	СД3	О2	СД3	СД2	О1	СД7
39.	СД3	О3	СД5	СД1	О1	СД6
40.	СД1	О2	СД3	СД1	О2	СД7
41.	СД2	О4	СД5	СД2	О4	СД7

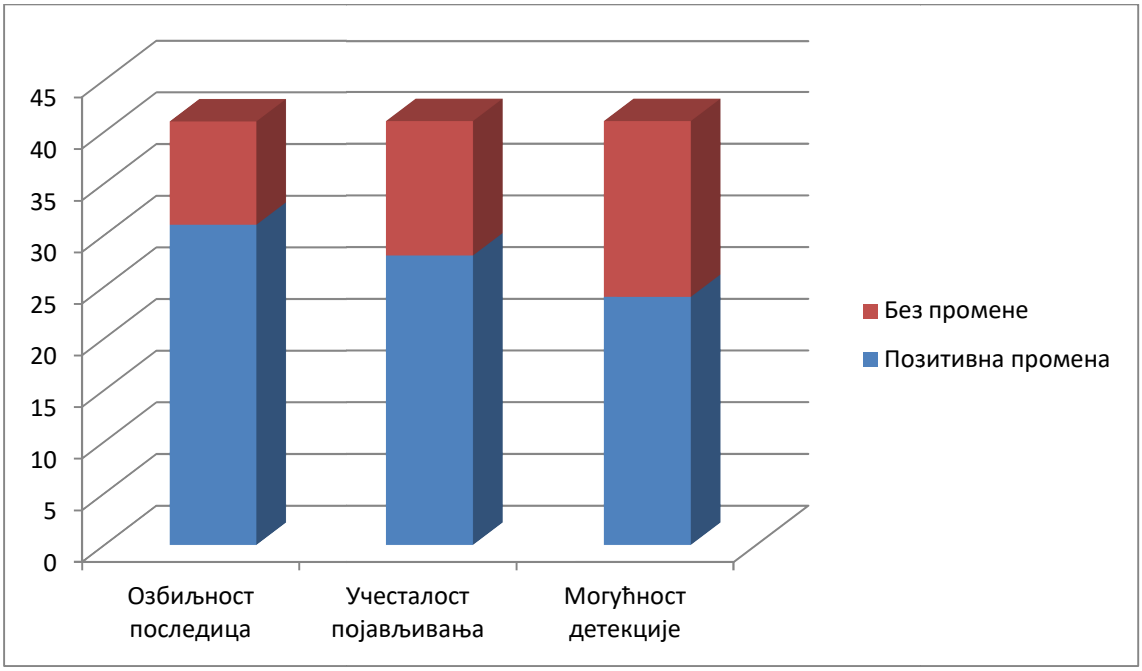
Када је у питању фактор ризика "Озбиљност последица" од 41 идентификоване грешке, дошло је до побољшања у оценама за 31 грешку, а за 10 грешака није било промене оцене. За 31 грешку од укупно 41 анализиране грешке, што представља 75,61%, дошло је до смањења процене озбиљности потенцијалних последица. Број грешака које су после примене мера и довољно дугог времена рада тако модификованог система, оцењене као грешке са најмањом могућом оценом из скале за оцену озбиљности последица "Без последица на проток података (СД1)" је тачно 16 грешака. То значи да после примене мера за 39,02% свих идентификованих грешака практично неће бити никаквих последица у случају материјализације грешке. Остали фактори ризика за те грешке више немају стварни значај и утицај на проток података. Ни једна грешка више нема оцену за овај фактор ризика изнад оцене СД3 што говори да је за све грешке ниво потенцијалне озбиљности последица сведен на мали утицај на проток података у најгорем случају.

За фактор ризика "Учесталост појављивања" дошло је до побољшања за 28 грешака, а за 13 није било промене. За 28 грешака од укупно 41 анализиране

грешке, што представља 68,29%, дошло је до смањења процене учесталости појављивања. Број грешака које су после примене мера оцењене као грешке са најмањом могућом оценом из скале за оцену учесталости појављивања "Веома ретко (О1)" је тачно 17, што је 41,46% свих идентификованих грешака.

За фактор ризика "Могућност детекције" од 41 идентификоване грешке, дошло је до побољшања у оценама за 24 грешке, а за 17 грешака није било промене оцене. За 24 грешке од укупно 41 анализиране грешке, што представља 58,54%, дошло је до повећања процене могућности детекције. Број грешака које су после примене мера оцењене као грешке са највећом могућом оценом из скале за оцену могућности детекције "Апсолутна ефикасност детекције (СД7)" је тачно 21 грешка, што је 51,22% свих идентификованих грешака.

Модификација информационог система у ужем смислу, спроведена је у облику унапређења софтвера кроз нову итерацију развоја. Из приказаних података види се да иако је остварен напредак у оценама свих фактора ризика, тај напредак није једнак. Највећи број позитивних промена остварен је на фактору ризика "Озбиљност последица", нешто мањи број на фактору "Учесталост појављивања" а најмањи број позитивних промена остварен је на фактору "Могућност детекције". На слици 28, приказан је удео позитивних промена фактора ризика у односу на укупан број анализираних грешака. Највећи број максимално добрих оцена после примењених мера међутим налази се у оценама фактора ризика "Могућност детекције". Брејнсторминг сесија софтверских стручњака на ову тему довела је до става да је овакав резултат очекиван и да је објашњење у природи грешака софтверских система. Као што је већ констатовано, софтвер се разликује од материјалних система. он је неопипљив, грешке софтвера зависе од динамичког понашања система. Стога је очекивано да ће дефинисане мере за отклањање грешака софтвера садржати елементе који ће довести до маскималног повећања способности детекције услова за настајања и последица грешака, квалитета детекције, тамо где то буде могуће. Разлог за релативно мање укупно смањење оцене учесталости настајања грешака је то што је део узрока за испољавање грешака ван софтверског система, па унапређења софтвера нису довољно утицала на укупан резултат.



Слика 28:Позитивне промене фактора ризика

Процене фактора ризика које су донете после примене мера, нису формиране са ослањањем на раније оцене. Осим ових нумеричких показатеља, интервјуисање стејкхолдера, крајњих корисника и развојног тима, показало је присуство перцепције о позитивној промени у функционисању система.

6. ЗАКЉУЧНА РАЗМАТРАЊА

Посматрајући првенствено трансакционе информационе системе, у докторској дисертацији предложена је нова методологија оптимизације ефеката протока података са интеграцијом CCA и анализе ризика у облику хибридне модификоване FMEA методе. Предложена методологија подразумева итеративни и инкрементални модел софтверског процеса као основу за успешну примену у пракси. Модификована хибридна FMEA метода која се посматрано у контексту софтверског процеса доменског софтвера, налази позиционирана у фази одржавања, као свој резултат даје списак мера које се примењују као активности модификације и наставка развоја софтверског решења. Неке од тако дефинисаних активности налазе се у фази дизајна, неке у фази имплементације дизајна, неке могу да се простиру на више фаза активности, али у сваком случају налазе се у ранијим фазама софтверског процеса, следећег циклуса. С друге стране, као улаз и предуслов за реализацију овакве предложене методологије, користе се елементи, посебно хијерархија декомпонованих процеса, спроведене CCA из ранијих фаза софтверског процеса из преходних итерација. Ова дуална међузависност, започиње условом иницијално спроведене структурне системске анализе.

У циљу повећања ефикасности протока података у трансакционим информационим системима, методологија предложена у докторској дисертацији базира се на анализи и процени грешака, као и на предузимању акционих мера у циљу њихове елиминације у процесима развоја и одржавања софтвера. Најчешће коришћена метода за анализу грешака у било којој организацији је FMEA метода. Уважавајући сугестије многих аутора да конвенционална FMEA метода има бројне недостатке, у овој дисертацији је предложен модел помоћу кога може на егзактан начин да се одреди ранг идентификованих грешака и одреди приоритет мера за отклањање грешака. У дисертацији је усвојено да доносиоци одлука у току спровођења анализе грешака уместо коришћења уобичајених нумеричких скала користе дефинисане лингвистичке исказе који су моделирани интервалним

тип 2 трапезоидним фази бројевима (IT2TrFN), сматрајући да фази скупови омогућавају да се неизвесности и непрецизности квантитативно опишу на одговарајући начин. Осим наведеног, у дисертацији је такође усвојено да се релативна важност РФ-а је задаје помоћу фази матрице парова упоређења са интервалним тип-2 трапезоидним фази бројевима, за разлику од конвенционалне методе где фактори ризика имају једнаку важност. Одређивање приоритета идентификованих грешака респектујући факторе ризика као и њихове тежине постављено је као задатак више-критеријумске оптимизације (ВКО) а ранг идентификованих грешака добија се применом конвенционалне COPRAS методе.

Експериментално истраживање

Предложена методологија је тестирана у пракси на активном софтверу и на стварним подацима са Машинског факултета Универзитета у Београду. Посматрани софтвер је на почетку истраживања био у употреби више од две године и већина иницијалних функционалних захтева је у том тренутку била решена. Из праксе се зна да не треба очекивати да све грешке софтвера буду елиминисане у току развоја, тестирања и инсталације. Најпре су дефинисани индикатори процеса информационог система који су праћени и реализована је софтверска имплементација потребних механизама евидентирања и праћења. Анализа ризика је спроведена према моделу у оквиру модификоване FMEA методе. Респектујући утврђени релативни значај грешака и дефинисане мере за отклањање грешака, спроведен је низ активности на побољшању и доради софтверског решења. Резултат реализоване методологије показује да је побољшана ефикасност свих праћених процеса информационог система са аспекта протока података, или у 48 од 52 мерена индикатора, што је 92%.

У току спроведеног експерименталног истраживања, утврђен је значај дефинисаног редоследа решавања узрока грешака. Међусобна условљеност промена у елементима софтверског решења значајно утиче на укупан резултат модификације и дораде у зависности од редоследа активности. Одређен број

идентификованих грешака није раније детектован уобичајеним приступом тестирању софтвера са симулацијама позитивних сценарија коришћења система.

Успех методологије у великој мери зависи од искуства укључених софтверских стручњака и од квалитета спроведене ССА. Неопходно је коришћење документованих софтверских захтева и дизајна система.

Дефинисани индикатори за праћење учинка процеса обраде података и софтверска имплементација за ову намену, могу бити уграђени директно у основну софтверску апликацију. На тај начин би константно праћење индикатора било једноставно, а кроз визуелни приказ олакшано тумачење.

За практичну примену приказане методологије, као што је претпостављено у фази теоријског разматрања а у експерименталној провери потврђено, неопходна је софтверска подршка.

Анализа потврђености полазних хипотеза

Сумирањем резултата експерименталног истраживања констатовани су следећи закључци:

- Потврђена је основна хипотеза да се интеграцијом структурне системске анализе (SSA - Structured System Analysis) - методе за анализу и функционалну декомпозицију информационог система и FMEA (Failure Mode and Efects Analysis) - методе за идентификацију, анализу и оцену ризика, могу: идентификовати процеси протока података, извршити структурна декомпозиција процеса протока података, идентификовати индикатори за праћење ефеката протока, одредити јединице мере индикатора ефеката протока, идентификовати ограничења индикатора ефеката протока података и идентификовати ризици ефеката протока података за посматрани информациони систем.,

- Потврђена је додатна хипотеза истраживања да је могуће интеграцијом SSA и FMEA пројектовати методологију за управљање ефектима протока података у информационим системима,

- Потврђена је додатна хипотеза истраживања да се оптимизација ефеката протока података у информационим системима реализује применом пројектоване методологије за управљање ефекатима протока података базиране на интеграцији SSA и FMEA.

Основне предности представљеног модела и методологије за примену у управљању информационим системима, уједно представљају и остварене главне доприносе:

- Адекватни лингвистички искази којима се описују релативни значај и вредности фактора ризика за информационе системе су развијени.

- Моделовање постојећих неизвесности помоћу IT2TrFNs.

- Разматрани проблем описан је формалним језиком, што је омогућило одређивање ранга софтверских грешака на егзактан начин.

- Редослед примене мера које треба да доведу до отклањања идентификованих грешака је одређен у складу са установљеним рангом. На овај начин значајно се мање троше ресури и ефикасније спроводи поступак побољшања софтвера.

- Предложени поступак је флексибилан и омогућава измене:

(1) броја грешака које се анализирају,

(2) релативног значаја и вредности фактора ризика,

(3) облик функције припадности интервал тип 2 Фази бројева може бити лако инкорпориран у предложени модел, и

(4) може бити лако проширен за анализу других проблема одлучивања и управљања у различитим областима истраживања.

- Методологија тесно повезује фазе дизајна софтверског процеса са фазом одржавања, како би се кроз итеративни начин развоја ефикасно идентификовале грешке софтвера које нису раније детектоване.

Ограничења предложене методологија и предлози за даља истраживања

С обзиром на уочена одређена ограничења предложеног модела, предлози праваца за даља истраживања у области могу бити усмерени ка:

- субјективност у процени улазних података на основу којих се одређује ранг грешака,

- висок степен зависности од искуства и знања доносилаца одлука,

- неопходност сарадње са ауторима доменског софтверског решења,

- сложеност рачунања у неким фазама анализе грешака,

- такође, препозната је потреба и значај чувања непрецизних и непотпуних информација у класичним релационим базама података као последица употребе фази логике. Проучавање до сада остварених резултата на овом пољу и истраживање нових могућности једноставног и ефикасног коришћења фази података у релационим базама података и помоћу SQL језика, биће област даљих истраживања.

Осим наведених ограничења предложене методологије у предметној докторској дисертацији, предлози за даља истраживања могу се концентрисати и на друге правце који нису сагледани током истраживања, а односе се на:

- анализу софтверских грешака у различитим информационим системима,

- наставак развоја софтверских алата за подршку примени предложене методологије,

- проширењу модела у оквиру модификоване FMEA са увођењем РФ-а битних за софтверски процес.

7. ЛИТЕРАТУРА

[1] Lucas C.H. ,1994, Information System Concept for Management. Mc Graw Hill, New York. 17.

[2] Gartner (January 2018), Worldwide IT Spending Forecast, STAMFORD, Conn., January 16, 2018

[3] Edsger W. Dijkstra, Keynote address to be given on 1 March 1999 at the ACM Symposium on Applied Computing at San Antonio, TX

[4] McConnell, S., 1996, Rapid Development: Taming Wild Software Schedules, Microsoft Press, ISBN 978-1556159008

[5] Royce, Winston (1970), "Managing the Development of Large Software Systems" (PDF), Proceedings of IEEE WESCON, 26 (August): 1–9

[6] Thayer, R., Dorfman, M., Garr, D., 2002, Software Engineering: Volume 1: The Development Process, Wiley-IEEE Computer Society, ISBN 978-0769515557

[7] Jacobson, I., Booch, G., Rumbaugh, J., 1999, The Unified Software Development Process, Addison-Wesley Professional, ISBN 978-0201571691

[8] Larman, C., 2003, Agile and Iterative Development: A Manager's Guide, Addison-Wesley Professional, ISBN: 978-0131111554]

[9] Boehm B, "A Spiral Model of Software Development and Enhancement", ACM SIGSOFT Software Engineering Notes, ACM, 11(4):14-24, August 1986

[10] Beck, K., Andres, C., 2004, Extreme Programming Explained: Embrace Change, Addison Wesley, ISBN 978-0321278654

[11] Yourdon E., Constantine L. (1978). Structured Design – Fundamentals of a Discipline of Computer Program and Systems Design, 2nd ed, Yourdon Press, New York. 1978.

[12] DeMarcoT. (1978). Structured Analysis and System Specification, Yourdon Press Computing Series, Prantice Hall, New Jersey

[13] D. Litchfield, C. Anley, J. Heasman, and B. Grindlay, The Database Hacker's Handbook, Wiley Publishing Inc., USA, 2005.

[14] Oracle Developer Community https://community.oracle.com/community/developerlsearch.jspa?peopleEnabled=true&userID=&containerType=&container=&q=select+trigger

[15] Fabbri, D., Ramamurthy, R. & Kaushik, R. (2013). SELECT triggers for data auditing. Proceedings of the 29th International Conference on Data Engineering (lCDE). IEEE: 1141-1152

[16] Qiang Huang, Lianzhong Liu, A Logging Scheme for Database Audit, Second International Workshop on Computer Science and Engineering, 2009

[17] https://en.wikipedia.org/wiki/List_of_Unified_Modeling_Language_tools

[18] Dias, Kapila. (2017). Evolvement of Computer Aided Software Engineering (CASE) Tools: A User Experience. International Journal of Computer Science and Software Engineering (IJCSSE). 6. 2409-4285.

[19] Avison, D.E., and Fitzgerald, G. Information Systems Development: Methodologies, Techniques and Tools, (Second edition. ed.) McGraw-Hill, Maidenhead, 1995.

[20] A. Fuggetta, "A classification of CASE technology," in Computer, vol. 26, no. 12, pp. 25-38, Dec. 1993, doi: 10.1109/2.247645.

[21] Đurić G., Mitrović Č., Vorotović G., Blagojević I., Vasić M.: Developing Self-Modifying Code Model, Journal of Applied Engineering Science, ISSN 1451-4117, Vol. 14, No. 2, 2016, pp. 239- 247.

[22] Stamatis, D. H. (2003). Failure mode and effect analysis: FMEA from theory to execution. ASQ Quality Press.

[23] Sharma, R. K., Kumar, D., & Kumar, P. (2005). Systematic failure mode effect analysis (FMEA) using fuzzy linguistic modelling. International Journal of Quality & Reliability Management, 22(9), 986-1004.

[24] Banduka, N., Tadić, D., Mačužić, I., & Crnjac, M. (2018). Extended process failure mode and effect analysis (PFMEA) for the automotive industry: The FSQC-PFMEA. Advances in Production Engineering & Management, 13(2), 206-215.

[25] Vladimir M. Popović, Branko M. Vasić, Branislav B. Rakićević, Goran S. Vorotović: „Optimisation of maintenance concept choice using risk-decision factor – a case study“; International Journal of Systems Science, Taylor & Francis, 2011

[26] ISO/TS 16949:2009. (2009). Quality management systems – Particular requirements for the application of ISO 9001:2008 for automotive production and relevant service part organizations (second revision), Bureau of Indian Standards, New Delhi.

[27] Patel, S. C., Graham, J. H., & Ralston, P. A. (2008). Quantitatively assessing the vulnerability of critical information systems: A new method for evaluating security enhancements. International Journal of Information Management, 28(6), 483-491.

[28] Peko, M., Komatina, N., Banduka, N., & Crnjac, M. [2018]. Ocena i rangiranje grešaka u industriji informacionih tehnologija zasnovani na FMEA i višekriterijumskoj optimizaciji. Ekonomski horizonti, 20(3), 257-268.

[29] Dedrick, J., Kraemer, K. L., & Shih, E. (2013). Information technology and productivity in developed and developing countries. Journal of Management Information Systems, 30(1), 97-122.

[30] Das Adhikary, D., Kumar Bose, G., Bose, D., & Mitra, S. (2014). Multi criteria FMECA for coal-fired thermal power plants using COPRAS-G. International Journal of Quality & Reliability Management, 31(5), 601-614.

[31] Liu, H. C., You, J. X., & You, X. Y. (2014). Evaluating the risk of healthcare failure modes using interval 2-tuple hybrid weighted distance measure. Computers & Industrial Engineering, 78, 249-258.

[32] Liu, H. C., Li, P., You, J. X., & Chen, Y. Z. (2015). A Novel Approach for FMEA: Combination of Interval 2-Tuple Linguistic Variables and Gray Relational Analysis. Quality and Reliability Engineering International, 31(5), 761-772.

[33] Liu, H. C., Fan, X. J., Li, P., & Chen, Y. Z. (2014). Evaluating the risk of failure modes with extended MULTIMOORA method under fuzzy environment. Engineering Applications of Artificial Intelligence, 34, 168-177.

[34] Liu, H. C., You, J. X., You, X. Y., & Shan, M. M. (2015). A novel approach for failure mode and effects analysis using combination weighting and fuzzy VIKOR method. Applied Soft Computing, 28, 579-588.

[35] Song, W., Ming, X., Wu, Z., & Zhu, B. (2014). A rough TOPSIS approach for failure mode and effects analysis in uncertain environments. Quality and Reliability Engineering International, 30(4), 473-486.

[36] Meng Tay, K., & Peng Lim, C. (2006). Fuzzy FMEA with a guided rules reduction system for prioritization of failures. International Journal of Quality & Reliability Management, 23(8), 1047-1066.

[37] Dubois, D., & Prade, H. (1998). An introduction to fuzzy systems1. Clinica Chimica Acta, 270(1), 3-29.

[38] Zimmermann, H. J. (2010). Fuzzy set theory. Wiley Interdisciplinary Reviews: Computational Statistics, 2(3), 317-332.

[39] Liu, H. C., Liu, L., Liu, N., & Mao, L. X. (2012). Risk evaluation in failure mode and effects analysis with extended VIKOR method under fuzzy environment. Expert Systems with Applications, 39(17), 12926-12934.

[40] Tadic, D., Aleksic, A., Popovic, P., Arsovski, S., Castelli, A., Joksimovic, D., and Stefanovic, M.: The evaluation and enhancement of quality, environmental protection and seaport safety by using FAHP, Nat. Hazards Earth Syst. Sci., 17, 261–275, https://doi.org/10.5194/nhess-17-261-2017, 2017.

[41] Liu, H. C., You, J. X., & Duan, C. Y. (2017). An integrated approach for failure mode and effect analysis under interval-valued intuitionistic fuzzy environment. International Journal of Production Economics.

[42] Liu, H. C., You, J. X., Li, P., & Su, Q. (2016). Failure Mode and Effect Analysis Under Uncertainty: An Integrated Multiple Criteria Decision Making Approach. IEEE Trans. Reliability, 65(3), 1380-1392.

[43] Zadeh, L. A. (1975). The concept of a linguistic variable and its application to approximate reasoning—I. Information sciences, 8(3), 199-249.

[44] Mendel, J. M. (2017). Type-2 fuzzy sets. In Uncertain Rule-Based Fuzzy Systems, 259-306. Springer, Cham.

[45] Đurić Goran P, Mitrović Časlav B, Komatina Nikola, Tadić Danijela P, Vorotović Goran S: „The hybrid MCDM model with the interval Type-2 fuzzy sets for the software failure analysis“; JOURNAL OF INTELLIGENT & FUZZY SYSTEMS, 2019, DOI:10.3233/JIFS-182541.

[46] Tadic, D., Aleksic, A., Stefanovic, M., Arsovski, S., 2014. Evaluation and Ranking of Organizational Resilience Factors by Using a Two-Step Fuzzy AHP and Fuzzy TOPSIS. Math. Probl. Eng. 418085(13 pp.).

[47] Nestic, Snezana; Djordjevic, Aleksandar; Puskaric, Hrvoje; Djordjevic, Marija Zahar; Tadic, Danijela; Stefanovic, Miladin,Jan 01 2015,IOS Press,The evaluation and improvement of process quality by using the fuzzy sets theory and genetic algorithm approach, journal-of-intelligent-and-fuzzy-systems/ifs1679

[48] Kutlu, A. C., & Ekmekçioğlu, M. (2012). Fuzzy failure modes and effects analysis by using fuzzy TOPSIS-based fuzzy AHP. Expert Systems with Applications, 39(1), 61-67.

[49] Chang, D. Y. (1996). Applications of the extent analysis method on fuzzy AHP. European journal of operational research, 95(3), 649-655.

[50] Sachdeva, A., Kumar, D., & Kumar, P. (2009). Multi-factor failure mode critically analysis using TOPSIS.

[51] Abdullah, L., & Najib, L. (2014). A new type-2 fuzzy set of linguistic variables for the fuzzy analytic hierarchy process. Expert Systems with Applications, 41(7), 3297-3305.

[52] Kahraman, C., Öztayşi, B., Sarı, İ. U., & Turanoğlu, E. (2014). Fuzzy analytic hierarchy process with interval type-2 fuzzy sets. Knowledge-Based Systems, 59, 48-57.

[53] Zavadskas, E. K., & Kaklauskas, A. (1996). Determination of an efficient contractor by using the new method of multicriteria assessment. In International Symposium for "The Organization and Management of Construction". Shaping Theory and Practice(Vol. 2, pp. 94-104).

[54] Fouladgar, M. M., Yazdani-Chamzini, A., Lashgari, A., Zavadskas, E. K., & Turskis, Z. (2012). Maintenance strategy selection using AHP and COPRAS under fuzzy environment. International journal of strategic property management, 16(1), 85-104.

[55] Ghorabaee, M. K., Amiri, M., Sadaghiani, J. S., & Goodarzi, G. H. (2014). Multiple criteria group decision-making for supplier selection based on COPRAS method with interval type-2 fuzzy sets. The International Journal of Advanced Manufacturing Technology, 75(5-8), 1115-1130.

[56] Turanoglu Bekar, E., Cakmakci, M., & Kahraman, C. (2016). Fuzzy COPRAS method for performance measurement in total productive maintenance: a

comparative analysis. Journal of Business Economics and Management, 17(5), 663-684.

[57] Opricovic, S., & Tzeng, G. H. (2004). Compromise solution by MCDM methods: A comparative analysis of VIKOR and TOPSIS. European journal of operational research, 156(2), 445-455.

[58] Chen, C. T. (2000). Extensions of the TOPSIS for group decision-making under fuzzy environment. Fuzzy sets and systems, 114(1), 1-9.

[59] Torfi, Fatemeh & Zanjirani Farahani, Reza & Rezapour, Shabnam. (2010). Fuzzy AHP to determine the relative weights of evaluation criteria and Fuzzy TOPSIS to rank the alternatives. Applied Soft Computing. 10. 520-528. 10.1016/j.asoc.2009.08.021.

[60] Rezaei, J., 2015. Best-worst multi-criteria decision-making method. Omega 53, 49e57.

[61] Г.Ђурић, М.Мисита, *Пројектовање информационог система студентске службе*, XI Скуп привредника и научника СПИН'17 - зборник радова, Београд, 09-10. новембар 2017.године, стр. 163-170.

[62] Lazarević B., Marjanović Z., Aničić N., Babarogić S.: Baze podataka, FON, Beograd, 2003., str. 351

[63] Rosziati Ibrahim and Siow Yen Yen, (2011). A Formal Model for Data Flow Diagram Rules, ARPN Journal of Systems and Software, Volume 1 No. 2, MAY 2011

[64] Pritchard, R. D., Roth, P. L., Jones, S. D., & Roth, P.G. (1990). Implementing feedback systems to enhance productivity: a practical guide. National Productivity Review

[65] Zur Muehlen, M., & Shapiro, R. (2010). Business Process Analytics. U J. vom Brocke & M. Rosemann (Editori), Handbook on Business Process Management

2: Strategic Alignment, Governance, People and Culture (str. 137-157). Berlin: Springer

[66] Franceschini, F., Galetto, M., & Maisano, D. (2007). Management by measurement - Designing key indicators and performance measurement systems. New York: Springer Berlin Heidelberg

[67] Dumond, E. J. (1994). Making Best Use of Performance Measures and Information International Journal of Operations & Production Management, 14(9), 16 - 31

[68] Bellman, R., Droemer, D., Lohmann, M., & Miller, C. (1994). Performance Measurement Process Guidance Document. Las Vegas: Department of Energy.

[69] S. Michael Groomer, Uday S. Murthy, 2 . "Continuous Auditing of Database Applications: An Embedded Audit Module Approach 1 " In Continuous Auditing. Published online: 12 Mar 2018; 105-124.

[70] Martin, James (1983). Managing the Data-base Environment. Englewood Cliffs, New Jersey: Prentice-Hall. p. 381.

[71] Do, N. (2014). Application of OLAP to a PDM database for interactive performance evaluation of in-progress product development. Computers In Industry, 65(4), 636-645. doi:10.1016/j.compind.2014.01.014

[72] Turban, E., Sharda, R., Delen, D., & King, D. (2010). Business intelligence: A managerial approach. New Jersey: Pearson Prentice Hall.

[73] TL Saaty, LG Vargas - Decision making with the analytic network process, 2013

[74] Runkler, Thomas & Coupland, Simon & John, Robert & Runkler@siemens, Thomas & Com,. (2016). Interval Type–2 Fuzzy Decision Making. International Journal of Approximate Reasoning. 80. 10.1016/j.ijar.2016.09.007.

[75] Qin, J., & Liu, X. (2015). Multi-attribute group decision making using combined ranking value under interval type-2 fuzzy environment. Information Sciences, 297, 293-315.

[76] Aleksic, A.; Runic Ristic, M.; Komatina, N.; Tadic, D. (2019). Advanced risk assessment in reverse supply chain processes: A case study in Republic of Serbia, Advances in Production Engineering & Management, Vol. 14, No. 4, 421-434, https://doi.org/10.14743/apem2019.4.338

[77] Macuzić, I., Tadić, D., Aleksić, A., & Stefanović, M. (2016). A two step fuzzy model for the assessment and ranking of organizational resilience factors in the process industry. Journal of Loss Prevention in the Process Industries, 40, 122-130.

[78] Johnson, K. G., & Khan, M. K. (2003). A study into the use of the process failure mode and effects analysis (PFMEA) in the automotive industry in the UK. Journal of Materials Processing Technology, 139(1), 348-356.

[79] Russomanno, D. J., Bonnell, R. D., & Bowles, J. B. (1994). Viewing computer-aided failure modes and effects analysis from an artificial intelligence perspective. Integrated Computer-Aided Engineering, 1(3), 209-228.

[80] Hunt, J. E., Pugh, D. R., & Price, C. J. (1995). Failure mode effects analysis: a practical application of functional modeling. Applied Artificial Intelligence an International Journal, 9(1), 33-44.

[81] Wirth, R., Berthold, B., Krämer, A., & Peter, G. (1996). Knowledge-based support of system analysis for the analysis of failure modes and effects. Engineering Applications of Artificial Intelligence, 9(3), 219-229.

[82] Huang, G. Q., & Mak, K. L. (2003). Failure mode and effect analysis (FMEA) over the WWW. In Internet Applications in Product Design and Manufacturing (pp. 135-146). Springer Berlin Heidelberg.

[83] Gan, L., Xu, J., & Han, B. T. (2012). A computer-integrated FMEA for dynamic supply chains in a flexible-based environment. The International Journal of Advanced Manufacturing Technology, 59(5-8), 697-717.

[84] Parnas, D. L. (1974). On a "Buzzword": Hierarchical Structure. In Proceedings of the IFIP Congress'74 (pp.336–339).

[85] Aulbach, S., Grust, T., Jacobs, D., Kemper, A., & Rittinger, J. (2008, June). Multi-tenant databases for software as a service: schema-mapping techniques. In Proceedings of the 2008 ACM SIGMOD international conference on Management of data (pp. 1195-1206).

[86] Г. Лазовић, Г. Воротовић, Ч. Митровић, И. Аранђеловић, А. Бенгин: „*Напредни алати за управљање базама података*"; Машински факултет 2017., ISBN-978-86-7083-953-3, 127 стр.

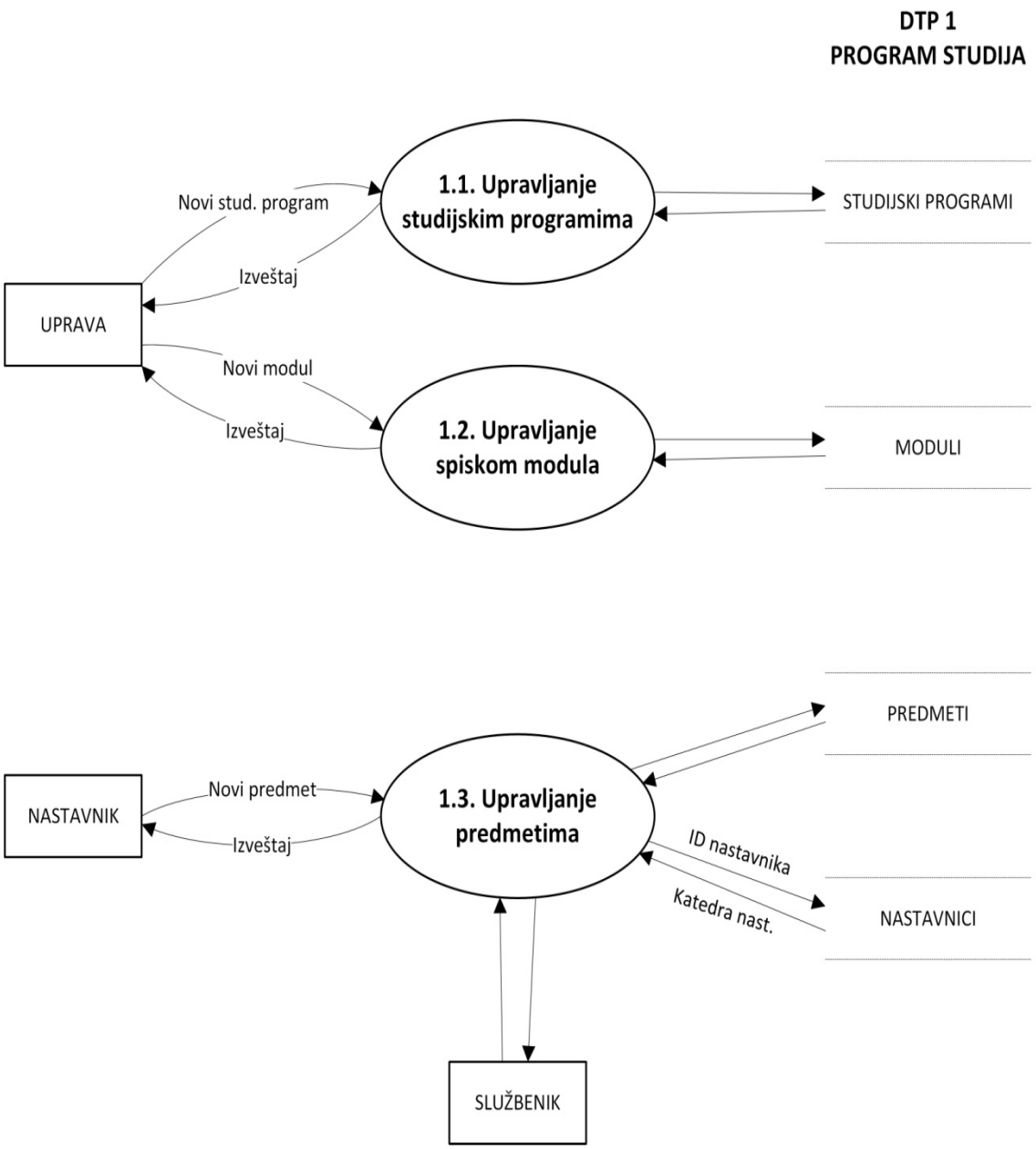
[87] Armbrust, Michael & Fox, Armando & Griffith, Rean & Joseph, Anthony & Katz, Randy & Konwinski, Andy & Lee, Gunho & Patterson, David & Rabkin, Ariel & Stoica, Ion & Zaharia, Matei. (2010). A View of Cloud Computing. Commun. ACM. 53. 50-58. 10.1145/1721654.1721672.

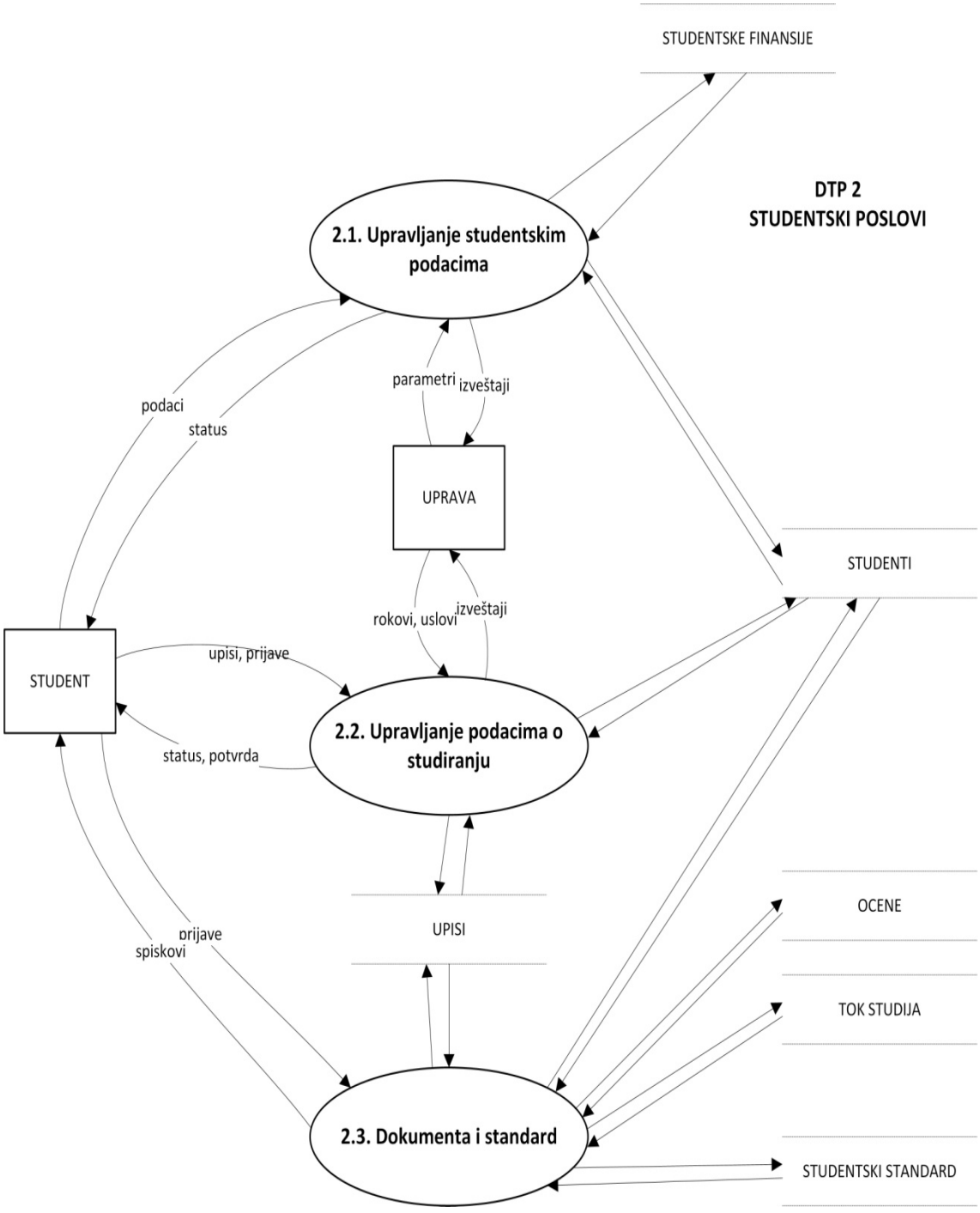
[88] S.M. McMenamin and J.F. Palmer, Essential systems analysis, Yourdon Press, New York, 1984.

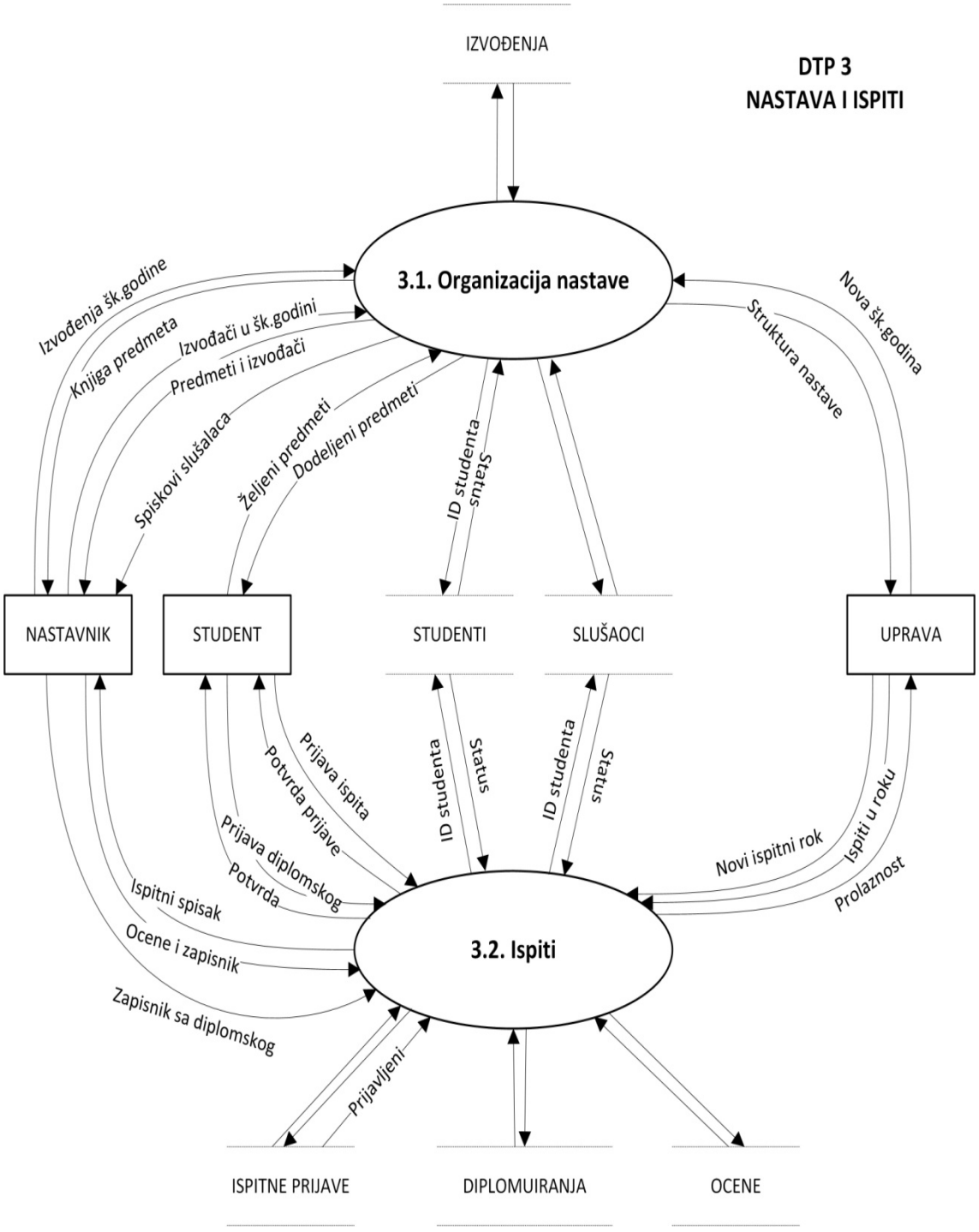
[89] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, (1995). Design patterns : elements of reusable object-oriented software. Reading, Mass. :Addison-Wesley

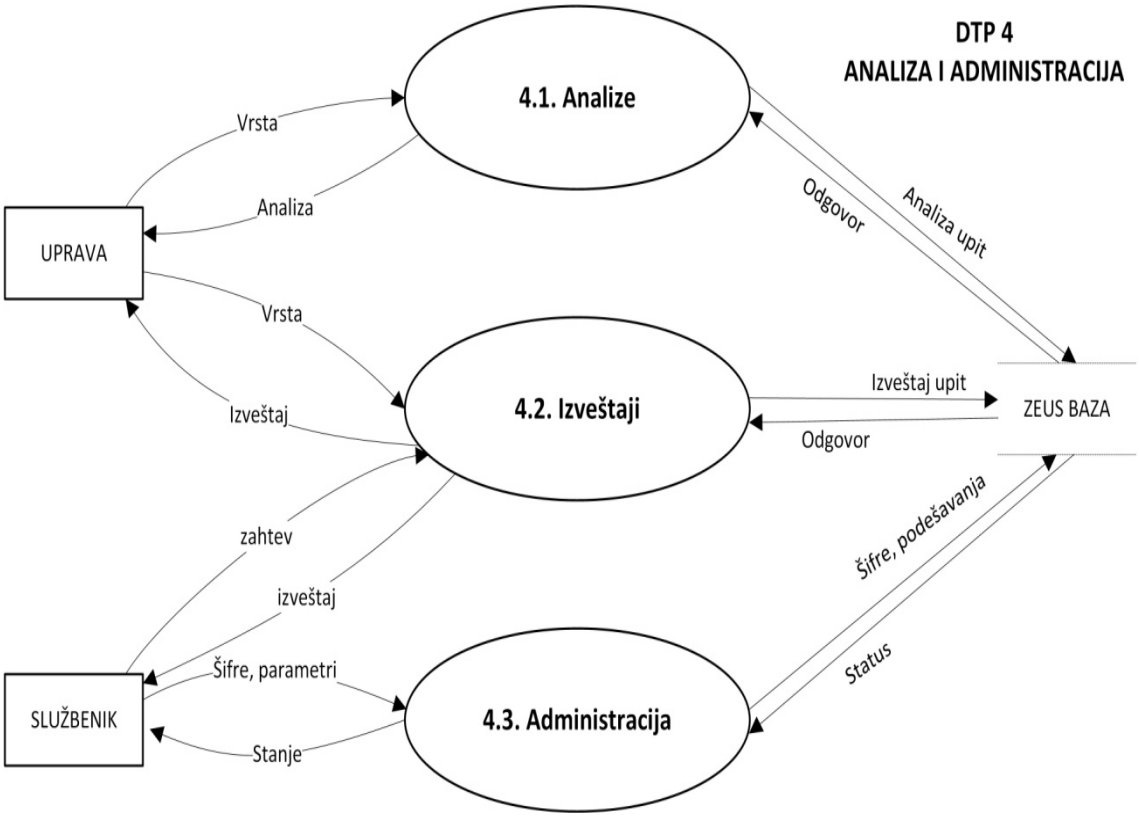
8. ПРИЛОЗИ

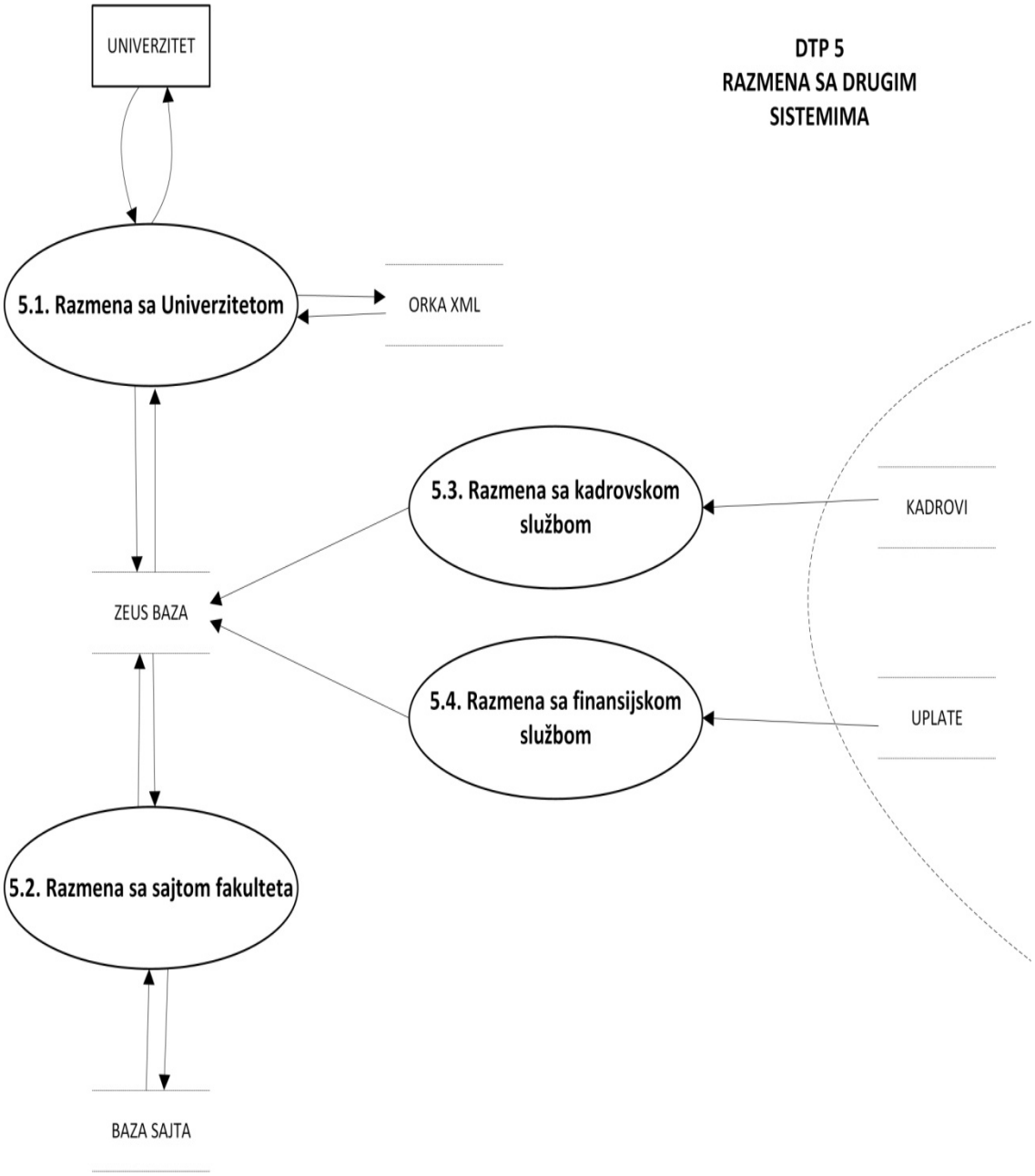
Дијаграми тока података првог ниоа информационог система ст.службе :




STUDENTSKE FINANSIJE
DTP 2
STUDENTSKI POSLOVI
2.1. Upravljanje studentskim podacima
parametri
izveštaji
podaci
status
UPRAVA
STUDENTI
rokovi, uslovi
izveštaji
STUDENT
upisi, prijave
2.2. Upravljanje podacima o studiranju
status, potvrda
UPISI
OCENE
prijave
spiskovi
TOK STUDIJA
2.3. Dokumenta i standard
STUDENTSKI STANDARD


IZVOĐENJA
DTP 3
NASTAVA I ISPITI
3.1. Organizacija nastave
Izvođenja šk.godine
Knjiga predmeta
Izvođači u šk.godini
Predmeti i izvođači
Spiskovi slušalaca
Željeni predmeti
Dodeljeni predmeti
ID studenta
Status
Struktura nastave
Nova šk.godina
NASTAVNIK
STUDENT
STUDENTI
SLUŠAOCI
UPRAVA
Prijava ispita
Potvrda prijave
Prijava diplomskog
Potvrda
ID studenta
Status
ID studenta
Status
Novi ispitni rok
Ispiti u roku
Prolaznost
Ispitni spisak
Ocene i zapisnik
Zapisnik sa diplomskog
3.2. Ispiti
Prijavljeni
ISPITNE PRIJAVE
DIPLOMUIRANJA
OCENE


DTP 4
ANALIZA I ADMINISTRACIJA
4.1. Analize
4.2. Izveštaji
4.3. Administracija
UPRAVA
SLUŽBENIK
ZEUS BAZA
Vrsta
Analiza
Vrsta
Izveštaj
zahtev
izveštaj
Šifre, parametri
Stanje
Odgovor
Analiza upit
Izveštaj upit
Odgovor
Šifre, podešavanja
Status


DTP 5
RAZMENA SA DRUGIM SISTEMIMA
UNIVERZITET
5.1. Razmena sa Univerzitetom
ORKA XML
5.3. Razmena sa kadrovskom službom
KADROVI
ZEUS BAZA
5.4. Razmena sa finansijskom službom
UPLATE
5.2. Razmena sa sajtom fakulteta
BAZA SAJTA

Списак свих процеса из ССА информационог система:

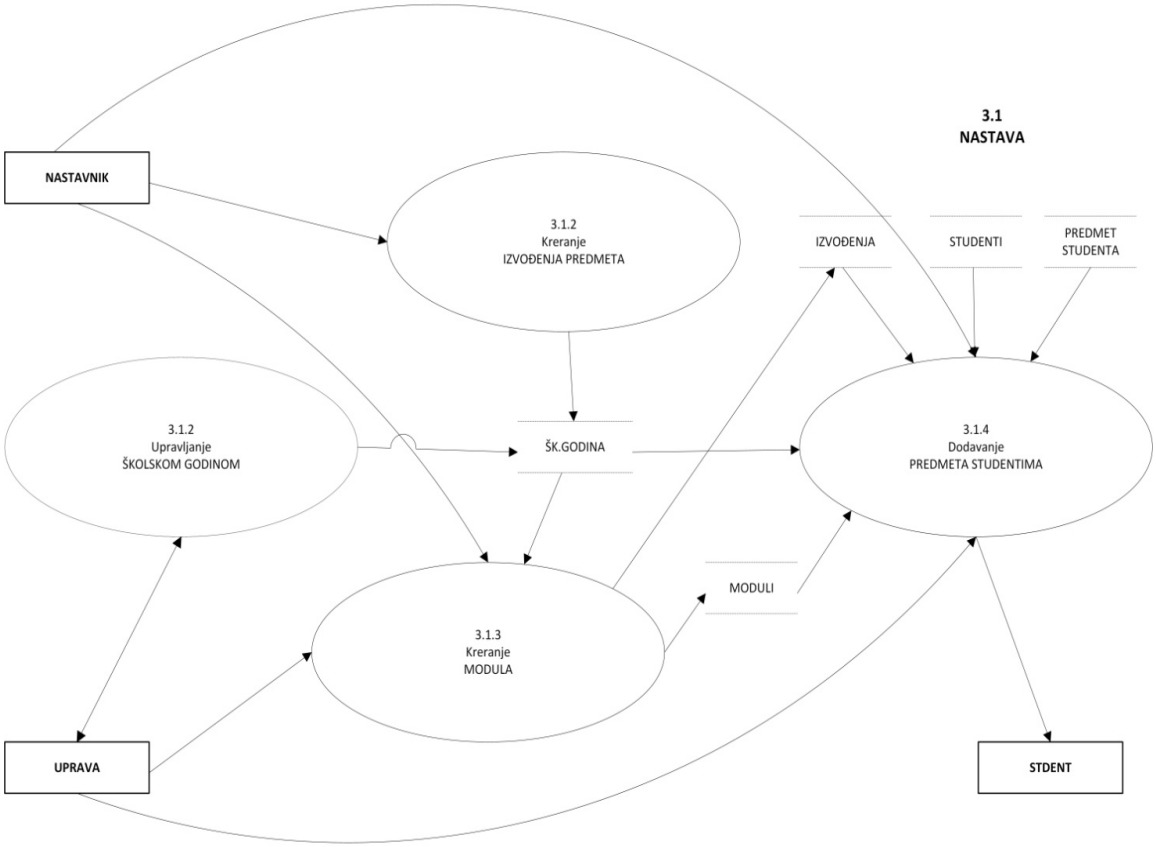
0.	Информациони систем Студентске службе
1.	Програм студија
1.1.	Управљање студијским програмима
1.2.	Управљање модулима
1.3.	Управљање предметима
1.3.1.	Креирање предмета
1.3.2.	Повезивање са акредитацијом
1.3.3.	Управљање еквиваленцијом
2.	Студентски послови
2.1.	Управљање студентским подацима
2.1.1.	Управљање матичним ст.подацима
2.1.1.1.	Управљање основним матичним ст.подацима
2.1.1.2.	Управљање соц-еко матичним ст.подацима
2.1.2.	Управљање финансијском евиденцијом
2.1.2.1.	Задужења студената
2.1.2.2.	Обрада уплата
2.1.2.3.	Финансијско извештавање
2.2.	Управљање подацима о студирању
2.2.1.	Упис на факултет
2.2.1.1.	Управљање уписним роковима
2.2.1.2.	Пријављивање кандидата
2.2.1.3.	Пријемни испит
2.2.1.4.	Формирање ранг листе
2.2.1.5.	Упис студената
2.2.2.	Управљање током студија
2.2.2.1.	Упис школске године
2.2.2.2.	Испис студента
2.2.2.3.	Дипломирање

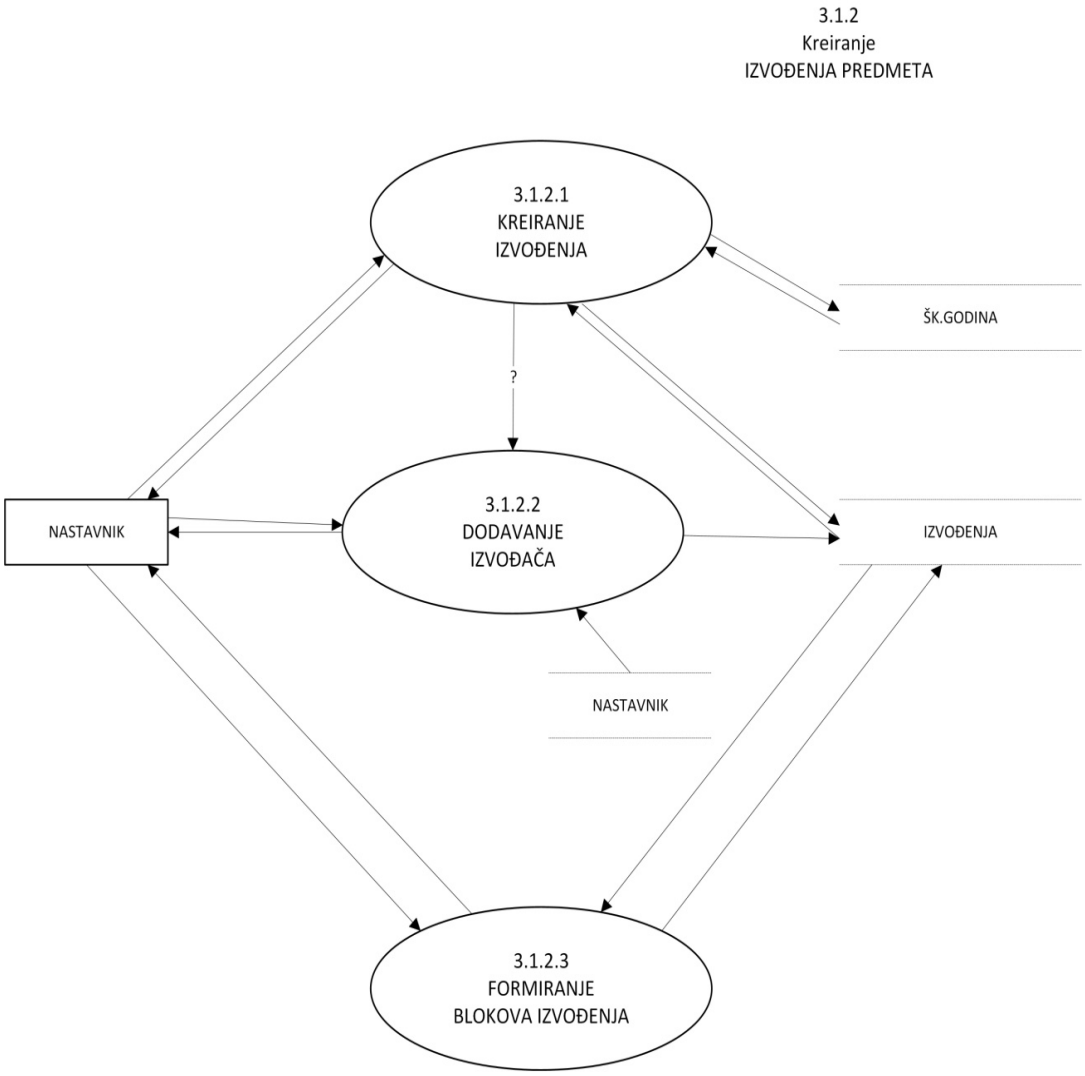
2.3.	Документа и стандард
2.3.1.	Израда докумената
2.3.1.1.	Уверење о дипломирању
2.3.1.2.	Потврда о статусу
2.3.1.3.	Уверење о положеним испитима
2.3.1.4.	Израда комплетног извештаја
2.3.2.	Студентски стандард
2.3.2.1.	Пријава за дом
2.3.2.2.	Формирање листе за дом
2.3.2.3.	Пријава за кредит/стипендију
2.3.2.4.	Формирање листе за кредит/стипендију
3.	Настава и испити
3.1.	Организација наставе
3.1.1.	Управљање школским годинама
3.1.2.	Извођења предмета
3.1.2.1.	Креирање извођења предмета
3.1.2.2.	Одређивање извођача предмета
3.1.2.3.	Креирање блокова предмета
3.1.3.	Модули
3.1.3.1.	Креирање студијских модула
3.1.3.2.	Одређивање обавезних предмета модула
3.1.3.3.	Одређивање изборних предмета модула
3.1.4.	Избор предмета студента
3.1.4.1.	Прикупљање и обрада жеља студената
3.1.4.2.	Додељивање предмета на основу жеља
3.1.4.3.	Директна додела предмета студенту
3.1.4.4.	Извештавање о додељеним предметима
3.2.	Испити и резултати
3.2.1.	Предиспитне обавезе
3.2.1.1.	Унос потписа

3.2.1.2.	Одрицање од потписа
3.2.1.3.	Извештавање
3.2.2.	Испити
3.2.2.1.	Управљање испитним роковима
3.2.2.2.	Управљање испитима
3.2.2.3.	Пријава испита
3.2.2.4.	Формирање испитних спискова
3.2.2.5.	Обрада резултата
3.2.2.6.	Анализа испита
3.2.3.	Дипломски радови
3.2.3.1.	Пријава дипломског
3.2.3.2.	Провера услова за дипломирање
3.2.3.3.	Евидентирање дипломирања
4.	Анализе и администрација
4.1.	Анализе
4.1.1.	Избор анализе
4.1.2.	Анализа пролазности
4.1.3.	Континуитет статуса студената
4.2.	Извештаји
4.2.1.	Избор извештаја
4.2.2.	Студентске финансије извештаји
4.2.3.	Студенти по годинама и предметима
4.2.4.	Дипломирани и дипломе
4.3.	Администрација
4.3.1.	Администрација дипломирања
4.3.2.	Администрација уговора
4.3.3.	Администрација шифарника
5.	Размена
5.1.	Размена са Универзитетом
5.1.1.	Импорт из ИСУ

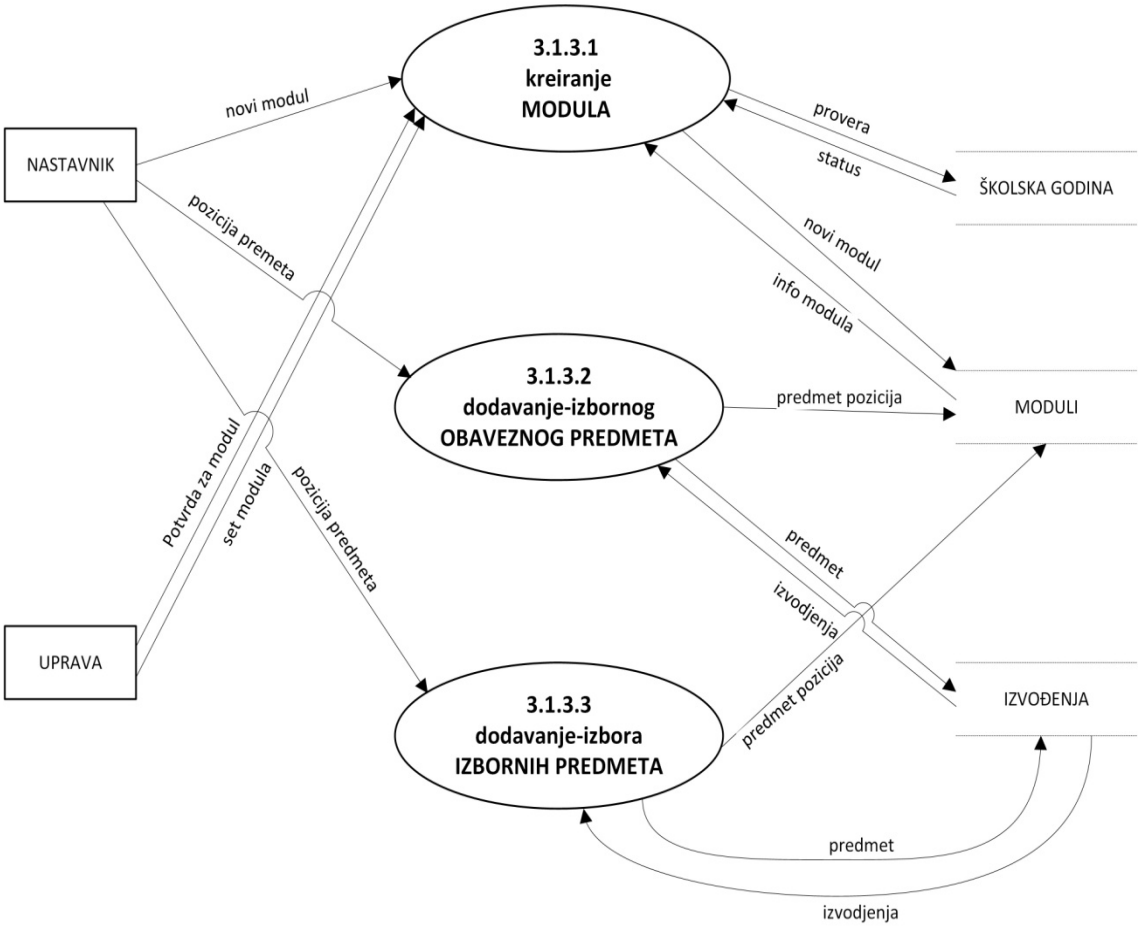
5.1.2.	Експорт ка ИСУ
5.2.	Размена са web сајтом факултета
5.2.1.	Импорт са студентског сервиса
5.2.2.	Експорт ка бази web сајта
5.3.	Размена са кадровском службом
5.4.	Размена са финансијском службом

Декомпоновани дијаграми из процеса 3:

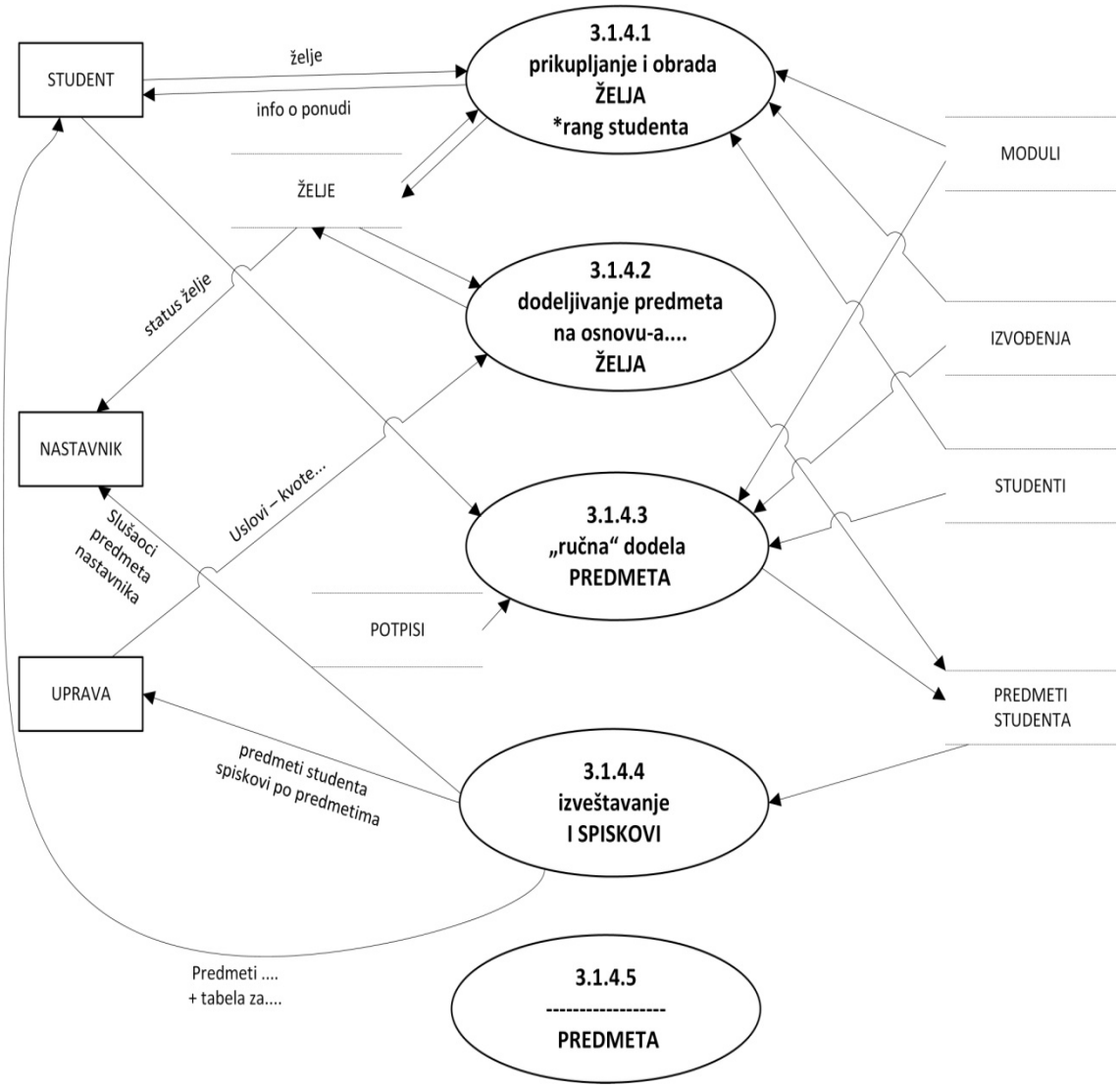


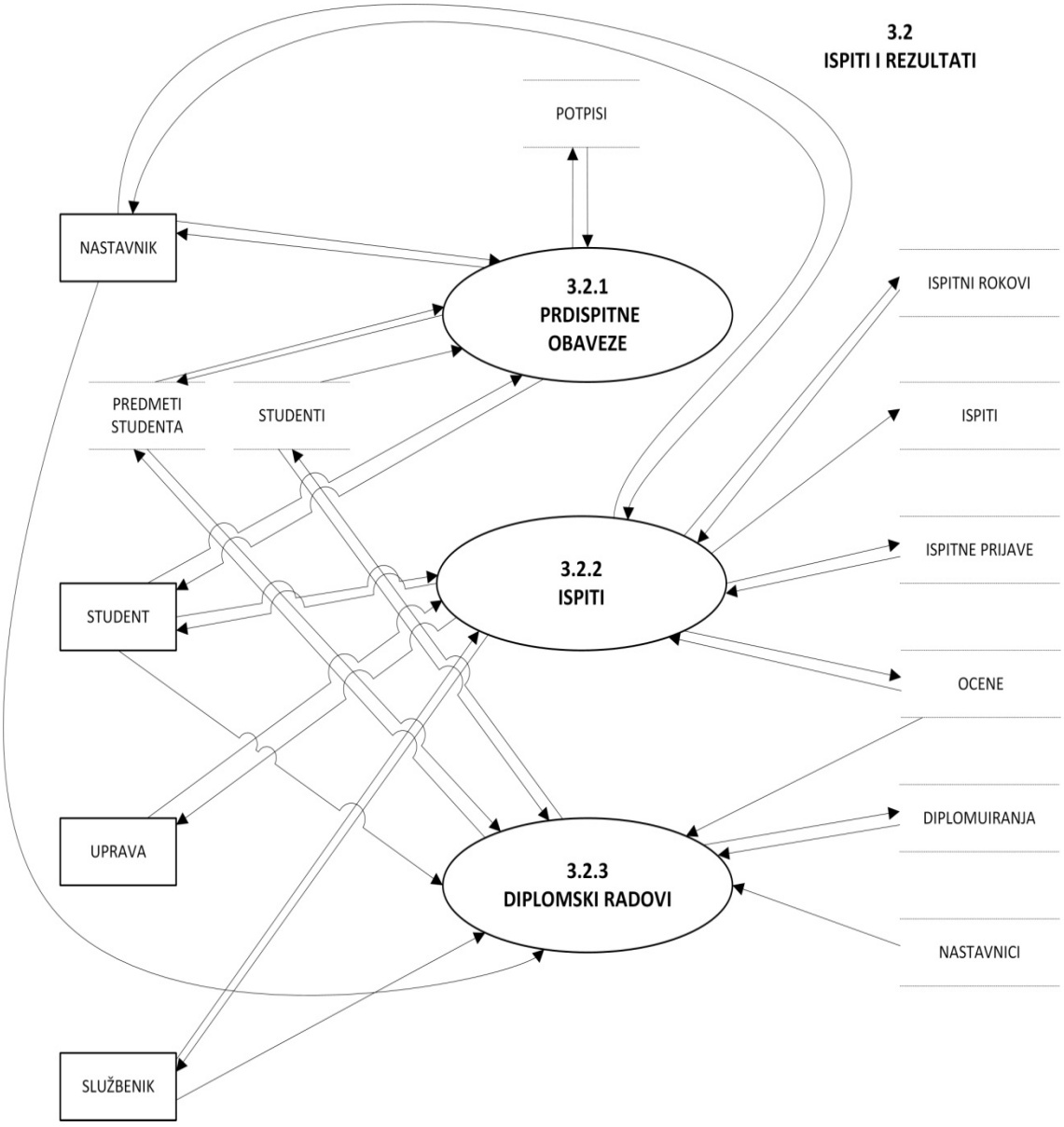

3.1.2
Kreiranje
IZVOĐENJA PREDMETA
3.1.2.1
KREIRANJE
IZVOĐENJA
ŠK.GODINA
?
NASTAVNIK
3.1.2.2
DODAVANJE
IZVOĐAČA
IZVOĐENJA
NASTAVNIK
3.1.2.3
FORMIRANJE
BLOKOVA IZVOĐENJA

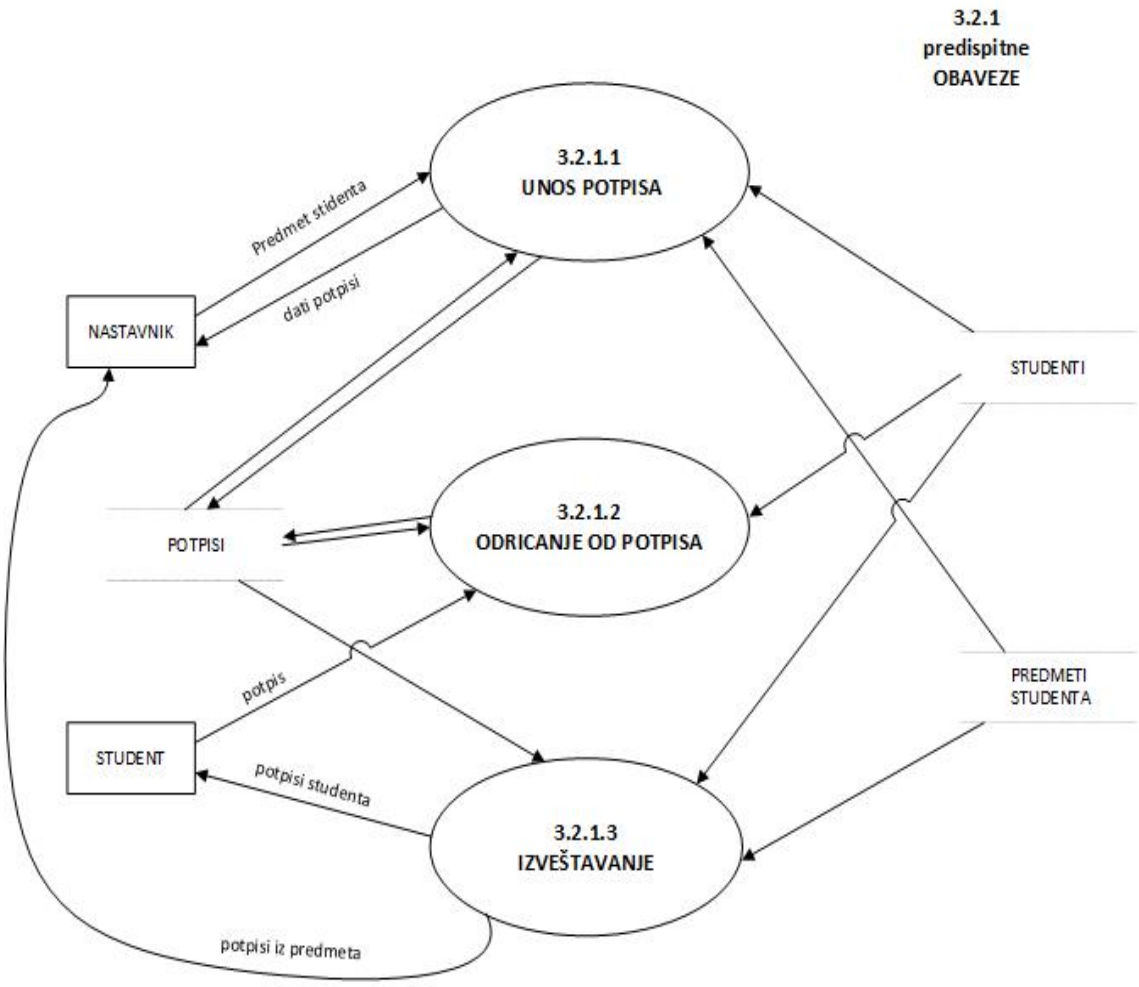
3.1.3.
Upravljanje MODULIMA

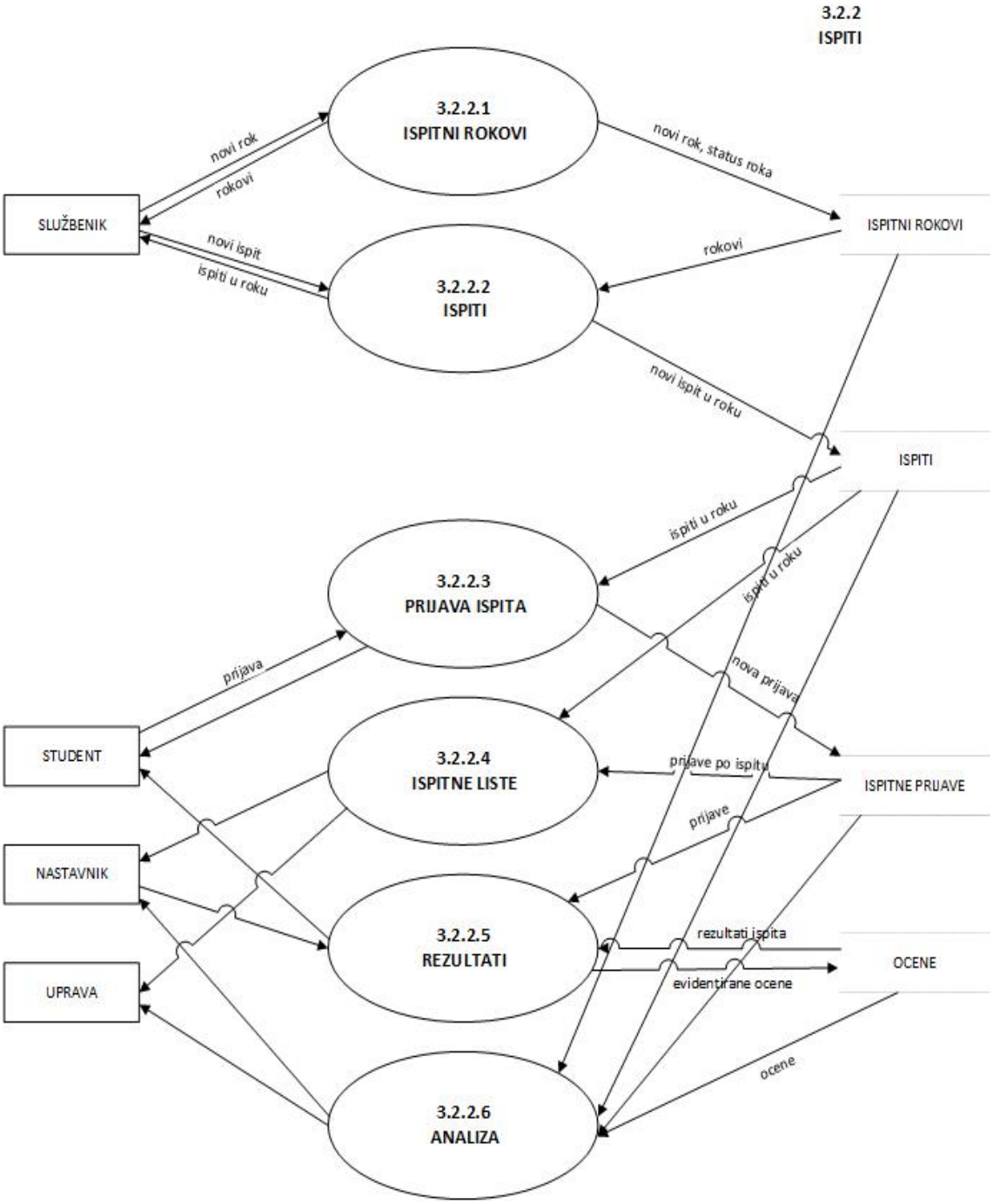


3.1.4.
dodeljivanje
PREDMETA STUDENTIMA

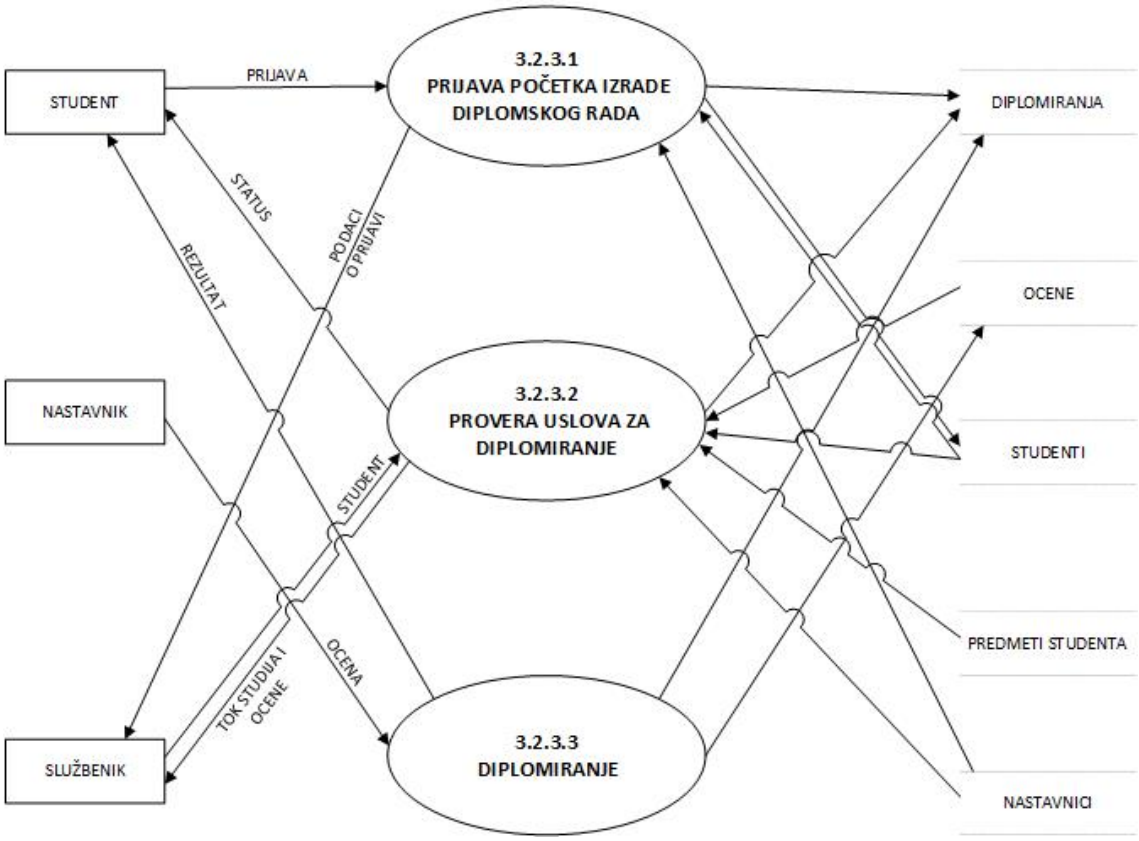



3.2
ISPITI I REZULTATI
POTPISI
NASTAVNIK
3.2.1
PRDISPITNE
OBAVEZE
ISPITNI ROKOVI
PREDMETI
STUDENTA
STUDENTI
ISPITI
3.2.2
ISPITI
ISPITNE PRIJAVE
STUDENT
OCENE
DIPLOMUIRANJA
UPRAVA
3.2.3
DIPLOMSKI RADOVI
NASTAVNICI
SLUŽBENIK


3.2.1
predispitne
OBAVEZE
3.2.1.1
UNOS POTPISA
Predmet studenta
dati potpisi
NASTAVNIK
STUDENTI
3.2.1.2
ODRICANJE OD POTPISA
POTPISI
PREDMETI
STUDENTA
potpis
STUDENT
potpisi studenta
3.2.1.3
IZVEŠTAVANJE
potpisi iz predmeta


3.2.2
ISPITI
3.2.2.1
ISPITNI ROKOVI
3.2.2.2
ISPITI
3.2.2.3
PRIJAVA ISPITA
3.2.2.4
ISPITNE LISTE
3.2.2.5
REZULTATI
3.2.2.6
ANALIZA
SLUŽBENIK
STUDENT
NASTAVNIK
UPRAVA
ISPITNI ROKOVI
ISPITI
ISPITNE PRIJAVE
OCENE
novi rok
rokovi
novi ispit
ispiti u roku
novi rok, status roka
rokovi
novi ispit u roku
ispiti u roku
ispiti u roku
prijava
nova prijava
prijave po ispitu
prijave
rezultati ispita
evidentirane ocene
ocene

3.2.3
DIPLOMSKI RADOVI



Биографски подаци о кандидату

Горан Ђурић је рођен 11.3.1968. год. у Паризу, Француска. У Шапцу је завршио основну школу, а потом и Шабачку гимназију. Завршио је Вишу школу за информатику у Београду 1992. године. Основне академске студије завршио је на Факултету информационих технологија у Београду 2009. године, са просеком 8.9. Мастер академске студије, модул Машинство и информационе технологије, завршио је на Машинском факултету Универзитета у Београду 2011. године са просеком 9.4. Докторске студије уписао је школске 2011/2012 године на Машинском факултету Универзитета у Београду. У току летњег семестра 2013. године као студент докторских студија је држао вежбе из предмета Софтверско инжењерство на модулу Машинство и информационе технологије.

Запослен је на Машинском факултету Универзитета у Београду као главни софтвер инжењер у Служби за студентске послове од јануара 2010. године до сада. Бави се развојем и одржавањем информационог система. За потребе Машинског факултета а према пројектном задатку добијеном од руководства факултета и дефинисаним захтевима од стране руководиоца Службе за студентске послове, пројектовао је и израдио софтверско решење за подршку процесима наставе и студирања на факултету и обраду припадајућих података. Имплементираним решењем остварена је интеграција токова података различитих делова информационог система факултета, обезбеђен приступ подацима из ранијих база података као и размена података са информационим системом Универзитета.

Претходно радно искуство је остварио на следећим пословима: 1992-1995.год. Водопривредно предузеће "Подриње", програмирање пословних апликација, 1996-1999.год. Амалтеја компјутерс, програмирање књиговодственог софтвера, 1999.год. Основна школа у Коцељеви, наставник информатике, 2000-2002.год. Уно Мартин група, пројектовање ИС-а, 2002-2006.год М-пласт доо, пројектовање ИС-а, систем администрација, 2007-2009.год. Interex - Intermarche, пројектовање и израда софтвера за обраду података.

Професионално користи следеће програмске језике, пакете и технологије: C, C++, C#, Java, VB, JavaScript, Fortran, PL1, Cobol, Pascal, Angular, Php, Asp, Html, Xml, Css, jQuery, Ajax, Node.js, Sql, Oracle, MS Sql server, MySql, PostgreSql, DB2, PL-Sql, Linq, Entity frameworks, .Net, Web API, SOA, WPF, WCF, UML, Design patterns.

Од страних језика говори енглески језик и француски језик.

Изјава о ауторству

Име и презиме аутора: Горан Ђурић

Број индекса: Д1/2011

Изјављујем

да је докторска дисертација под насловом:

МЕТОДОЛОГИЈА ОПТИМИЗАЦИЈЕ ЕФЕКАТА ПРОТОКА ПОДАТАКА НА БАЗИ ИНТЕГРАЦИЈЕ СТРУКТУРНЕ СИСТЕМСКЕ АНАЛИЗЕ И РИЗИКА

- резултат сопственог истраживачког рада;
- да дисертација у целини ни у деловима није била предложена за стицање друге дипломе према студијским програмима других високошколских установа;
- да су резултати коректно наведени и
- да нисам кршио/ла ауторска права и користио/ла интелектуалну својину других лица.

Потпис аутора

У Београду, 12.10.2020.

_____

Изјава о истоветности штампане и електронске верзије докторског рада

Име и презиме аутора: Горан Ђурић

Број индекса: Д1/2011

Студијски програм: Машинско инжењерство

Наслов рада: МЕТОДОЛОГИЈА ОПТИМИЗАЦИЈЕ ЕФЕКАТА ПРОТОКА ПОДАТАКА НА БАЗИ ИНТЕГРАЦИЈЕ СТРУКТУРНЕ СИСТЕМСКЕ АНАЛИЗЕ И РИЗИКА

Ментор: др Часлав Митровић, редовни професор

Изјављујем да је штампана верзија мог докторског рада истоветна електронској верзији коју сам предао/ла ради похрањивања у **Дигиталном репозиторијуму Универзитета у Београду.**

Дозвољавам да се објаве моји лични подаци везани за добијање академског назива доктора наука, као што су име и презиме, година и место рођења и датум одбране рада.

Ови лични подаци могу се објавити на мрежним страницама дигиталне библиотеке, у електронском каталогу и у публикацијама Универзитета у Београду.

Потпис аутора

У Београду, 12.10.2020.

_____

Изјава о коришћењу

Овлашћујем Универзитетску библиотеку „Светозар Марковић“ да у Дигитални репозиторијум Универзитета у Београду унесе моју докторску дисертацију под насловом:

МЕТОДОЛОГИЈА ОПТИМИЗАЦИЈЕ ЕФЕКАТА ПРОТОКА ПОДАТАКА НА БАЗИ ИНТЕГРАЦИЈЕ СТРУКТУРНЕ СИСТЕМСКЕ АНАЛИЗЕ И РИЗИКА

која је моје ауторско дело.
Дисертацију са свим прилозима предао/ла сам у електронском формату погодном за трајно архивирање.
Моју докторску дисертацију похрањену у Дигиталном репозиторијуму Универзитета у Београду и доступну у отвореном приступу могу да користе сви који поштују одредбе садржане у одабраном типу лиценце Креативне заједнице (Creative Commons) за коју сам се одлучио/ла.

1. Ауторство (CC BY)
2. Ауторство – некомерцијално (CC BY-NC)
3. Ауторство – некомерцијално – без прерада (CC BY-NC-ND)
4. Ауторство – некомерцијално – делити под истим условима (CC BY-NC-SA)
5. Ауторство – без прерада (CC BY-ND)
6. Ауторство – делити под истим условима (CC BY-SA)

Потпис аутора

У Београду, 12.10.2020.

_____

1. **Ауторство**. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце, чак и у комерцијалне сврхе. Ово је најслободнија од свих лиценци.
2. **Ауторство – некомерцијално**. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце. Ова лиценца не дозвољава комерцијалну употребу дела.
3. **Ауторство – некомерцијално – без прерада**. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, без промена, преобликовања или употребе дела у свом делу, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце. Ова лиценца не дозвољава комерцијалну употребу дела. У односу на све остале лиценце, овом лиценцом се ограничава највећи обим права коришћења дела.
4. **Ауторство – некомерцијално – делити под истим условима**. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце и ако се прерада дистрибуира под истом или сличном лиценцом. Ова лиценца не дозвољава комерцијалну употребу дела и прерада.
5. **Ауторство – без прерада**. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, без промена, преобликовања или употребе дела у свом делу, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце. Ова лиценца дозвољава комерцијалну употребу дела.

6. **Ауторство – делити под истим условима**. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце и ако се прерада дистрибуира под истом или сличном лиценцом. Ова лиценца дозвољава комерцијалну употребу дела и прерада. Слична је софтверским лиценцама, односно лиценцама отвореног кода.