Универзитет Св. „Кирил и Методиј”

Факултет за информатички науки и компјутерско инженерство

Докторска дисертација

Рамка за Развој на Научни Апликации

Автор:
м-р Бојана Котеска

Ментор:
вон. проф. д-р Анастас Мишев

Март, 2018
Скопје

Универзитет Св. „Кирил и Методиј”

Факултет за информатички науки и компјутерско инженерство

Апстракт

Рамка за Развој на Научни Апликации

од м-р Бојана Котеска

Постојаниот напредок во областа на природните науки несомнено ја наметнува потребата за развој на научни апликации со кои се врши симулација на феномените од природата. Недостатокот на користење на модели за развој на софтвер и практики на софтверското инженерство при развој на апликациите резултира во софтвер со слаб квалитет, проблеми во процесот на развој на софтверот, а подоцна и проблеми при рефакторирање на кодот, оптимизација и одржување.

Софтверското инженерство нуди големо множество на практики за секоја фаза од процесот на развој кои треба да се следат за да се создаде апликација според општо прифатените стандарди за квалитет на софтвер. Како резултат на специфичноста на овој тип апликации, стандардните препораки на софтверското инженерство не можат во целост да се применат, туку мора да се предложат соодветни модификации и прилагодувања на веќе постоечките софтверски процеси за развој.

Во оваа докторска теза се предлага рамка за развој на научни апликации која вклучува: модел за развој на научни апликации и модел за квалитет на научни апликации. Моделот за развој на апликации го дефинира процесот на развој на апликациите во фази, додека моделот за квалитет нуди можност за оценување на квалитетот на научната апликација преку атрибути и метрики. Главна цел на рамката е да се зголеми квалитетот на научните апликации и да се променат сегашните практики научниците.

Во докторската теза, исто така, се прикажани и техники на моделирање и развој на научни апликации кои опфаќаат примена на различни пристапи на развој и програмски модели, спецификација на барања, тестирање на софтвер, и сл.

Примената на оваа рамка се потврдува преку практичен пример, а дополнително се прикажува и компаративна анализа за оценување на квалитетот на две научни апликации според предложениот модел за квалитет. Користењето на рамката при развој на научните апликации резултира во организиран процес на развој во фази, специфицирани барања, дефинирани тест случаи, разбирлив код, автоматско тестирање, оценување на квалитетот на апликациите според веќе прифатени стандарди за квалитет, итн.

UNIVERSITY SS. CYRIL AND METHODIUS

Abstract

Faculty of Computer Science and Engineering

Framework for Developing Scientific Application

Bojana Koteska, MSc

The constant progress in the area of natural sciences, undoubtedly imposes the need for the development of scientific applications that simulate natural phenomena. The lack of using software development models and software engineering practices during the development process of software applications results in poor quality, problems in the development process and later problems with the code refactoring, optimization and maintenance.

Software Engineering offers a great set of practices for each stage of the development process that need to be followed in order create applications according to generally accepted standards for quality software. As a result of the specificity of this type of applications, the standard software engineering practices cannot be fully implemented, but appropriate modifications and adjustments to existing software development processes must be proposed.

In this doctoral thesis a framework for development of scientific applications is proposed including: scientific applications' development model and scientific applications' quality model. The application development model defines the process of developing applications in stages, while the quality model gives opportunity to assess the quality of scientific application through attributes and metrics. The main purpose of the framework is to increase the quality of scientific applications and to change current scientists' practices.

This doctoral thesis also presents scientific application modeling and developing techniques that include applying of the different approaches to development and programming models, requirements specification, software testing, etc.

The application of this framework is confirmed by a practical example, and additionally a comparative analysis for evaluating the quality of two scientific applications by using the proposed quality model is presented. The use of the framework in the development of scientific applications results in a development process organized in phases, specified requirements, defined test cases, understandable code, automatic testing, quality assessment of applications according to already accepted quality standards, etc.

Содржина

Листа на Слики

Листа на Табели

На моите родители...

Глава 1

Вовед

1.1 Мотивација и цели

Сознанијата од досегашните истражувања укажуваат дека при развој на научните апликации не се користат модели за развој на софтвер, ниту се следат практиките на софтверското инженерство со што се придонесува за намалување на квалитетот на научните апликации.

Како последица на овие сознанија произлегуваат неколку прашања и дилеми од кои се црпи главната мотивација на ова истражување: Која е причината за некористење на методите и практиките на софтверското инженерство? Кои се спецификите на научните апликации кои придонесуваат за користење на различен пристап кон развојот? Кои практики конкретно треба да се применат во секоја од фазите на развој на научната апликација и како тие можат да го подобрат квалитетот на научните апликации? Кои се последиците од неправилен развој на научни апликации? Какво е влијанието на принципите на софтверското инженерство врз коректноста на резултатите добиени со извршување на апликацијата? Што треба да се направи за да се променат досегашните навики за развој на научни апликации? Како да се подобри квалитетот на апликациите?

Предмет на ова истражување е дефинирањето на модел за развој на научни апликации и дефинирање на модел за квалитет. Пронаоѓањето на конкретни методи и стандарди за квалитет од софтверското инженерство кои може да се вклучат во процесот на развој на различните видови научни апликации, соодветно по фази, позитивно влијае на подобрување на развојниот процес и зголемувањето на квалитетот на апликациите.

Основната цел на ова истражување е да се подобри квалитетот на научните апликации преку користење на прифатени стандарди, практики и методи за развој на софтвер кои ги предлага софтверското инженерство, но секако и нивна соодветна модификација за да се вклопат поефикасно во развојниот процес на научните апликации. Откривањето на правилните методи кои може да се адаптираат во процесот на развој, бара детална анализа на карактеристиките на научните апликации, па една од целите на ова истражување, пред да се реализира главната цел, е да се воочат разликите и специфичностите на научните апликациите во однос на комерцијалните апликации за да може да се предложат подобри препораки.

Исто така, ова истражување се стреми кон подобрување на досегашните практики на развој на научниот софтвер преку заменување на лошите навики при развојот и занемарување на некои основни делови во фазите од развојот, како што се: создавањето на план за организација на активностите, дефинирање на барањата, дефинирање и креирање на тестови, документација и сл.

Научните апликации кои имаат поголем квалитет се многу поотпорни на грешките кои може да се појават во процесот на развој и со тоа се подобрува коректноста на крајните резултати, како и на процесот на верификација на резултатите добиени од извршената симулација.

1.2 Хипотези

Општа хипотеза:

Согледувањето на специфичностите на научните апликации, откривањето и адаптирањето на практиките кои ги нуди софтверското инженерство во процесот на развој, придонесува за значително зголемување на квалитетот на апликациите, не само преку подобрување на развојниот процес, туку и преку подобрување на процесот на верификација на добиените резултати.

Прва посебна хипотеза:

Воочувањето на разликите помеѓу научните апликации и комерцијалните апликации овозможува дефинирање на многу поспецифични методи и практики за научните апликации за секоја фаза од развојот пооделно.

Втора посебна хипотеза:

Правилното дефинирање на барања и тест случаи, постојаното тестирање, користејќи соодветни општо прифатени стандарди за тестирање на софтвер, како и навремено напишаната документација според одредени шаблони, се едни од клучните практики за подобрување на квалитетот на апликациите.

Трета посебна хипотеза:

Користењето на стандарди за квалитет и одредувањето на атрибути и метрики кои се значајни за квалитетот на научните апликации квантитативно ја потврдува придобивката од користењето на предложениот модел за развој на научни апликации.

1.3 Начин на истражување

Истражувањето ќе се спроведе користејќи неколку специфични методски постапки и техники на истражување: анализа, компарација, синтеза, индукција и дедукција. Исто така ќе се примени емпириска постапка за да се потврдат практиките и методите вклучени во рамката.

Најпрвин ќе се направи анализа на специфичностите на научните апликации, каде што ќе се согледаат карактеристиките и ќе се идентификуваат сегашните практики на развој и проблемите кои се јавуваат како резултат на тоа. Потоа, користејќи компарација ќе се направи преглед на сличностите и разликите меѓу комерцијалните и научните апликации. Компарацијата ќе помогне да се дефинираат поспецифични практики и методи во секоја од фазите на развој за различните типови на апликации.

Методот на индукција ќе се користи при претпоставката дека одредените методи кои што можат да се применат кај конкретна апликација ќе важат за сите апликации кои решаваат многу слични проблеми и припаѓаат на ист домен. Откако ќе се воочат можните примени на практитките од софтверското инженерство и стандардите за квалитет, ќе се примени техниката на синтеза, со чија помош ќе се креира рамка за квалитет на развој на научни апликации која ќе вклучи модел за развој на научните апликации и модел за квалитет.

Примената на дедуктивниот метод ќе овозможи коректно изведување на соодветни заклучоци за подобрување на процесот на развој на научните апликации.

Емпириската постапка ќе се примени преку практични примери на развој на научни апликации со што ќе се потврди корисноста на методите на софтверското инженерство при сите фази од развојниот процес на апликацијата, а со тоа и дека ќе се зголеми квалитетот на апликациите. Дополнително, со користење на моделот за квалитет, ќе се направи компарација помеѓу научна апликација развиена според предложениот модел за развој и научна апликација која се развива директно со програмирање на кодот.

1.4 Значење на истражувањето

Добиените резултати од истражувањето се очекува да дадат одговор на повеќе прашања, меѓу кои: Дали примената на софтверското инженерство може да го подобри развојниот процес и квалитетот на апликацијата?; Кои општо прифатени методи на софтверското инженерство можат да се вклучат во развојниот процес и какви модификации треба да се направат за нивна адаптација?; Дали постојат модели на развој на софтвер на кои може да се приспособи развојот на научните апликации?;

Одговорите на овие прашања ќе придонесат за менување на постоечкиот начин на развој на научни апликации каде што се занемаруваат практиките на софтверското инженерство, а со тоа се очекува и да се подобри организацијата на процесот и да се зголеми квалитетот.

Со решавање на овие прашања се очекува да се надминат досегашните недостатоци во развојот на научните апликации како што се: немањето на план за развој и документација, неправилното и ненавременото тестирање, не поседувањето документација и сл. Примената на овие чекори треба да го намали процентот на појавување на грешки, а доколку се појават грешки, тие да може да се коригираат што побрзо. Не поседувањето на соодветна документација, исто така, претставува проблем и кога некој софтверски инженер треба да се приклучи во развојниот процес, меѓутоа не може да го разбере кодот и

комплексната теорија од специфичната научна област, бидејќи не постои никаков документ за тоа.

Создавањето на научни апликации бара многу внимателно следење на процесот на развој и искуство, бидејќи симулациите кои се изведуваат може да имаат огромно значење за развојот на човештвото. Како пример за тоа може да се посочи решавањето на проблеми од областа на биомедицината и хемијата, кои вклучуваат и симулација на одредени експерименти чии резултати можат подоцна да се искористат за подобрување на здравјето на човекот и развој на фармацевтската индустрија. Неправилниот развој и лошата организација на активностите за развој не само што може да предизвикаат фатални последици по здравјето на човекот, туку исто така и загрозување на животната средина. Очекувањата од ова истражување се, исто така, да се спречат негативните последици, а и претходно да се воочат сите неправилности во развојниот процес.

Имајќи ја предвид комплексноста на проблемите од оваа област, се очекува дека зголемената организација на процесот на развој на научните апликации и следењето на одредени модели за развој и стандарди за квалитет ќе придонесе за поголем квалитет на софтверот и поефикасна искористеност на времето и ресурсите потребни за развој.

1.5 Структура

Докторската дисертација е организирана во повеќе поглавја.

Тезата започнува со поглавјето „Научни апликации" кое е организирано во повеќе подглави. Во првата подглава „Карактеристики на научните апликации" се прикажуваат општите карактеристики и дефиниции за научните апликации. Следно, во подглавата „Категоризација на научни апликации" се прави поделба на научните апликации според најзначајните карактеристики. Споредбата на карактеристиките на научните и комерцијалните апликации е прикажана во подглавата „Споредба на научни и комерцијални аппликации". Подглавата „Научниците и софтверскиот развој" е наменета за приказ на практиките на научниците кои ги применуваат при развој на научните апликации. Следните подглави од ова поглавје се однесуваат на поединечните фази од развојот на научните апликации: спецификација на барања, моделирање, кодирање, тестирање и управување со промени кај научните апликации.

Третото поглавје „Техники на моделирање и развој на научни апликации" е организирано во повеќе подглави, така што во секоја подглава се прикажува практичен пример од спроведено истражување. Во ова поглавје се прикажува примена на методи од софтверското инженерство и програмски модели за различни фази од развојот на научната апликација: спецификација на научни апликации, дизајн, препознавање и примена на програмски модели во фазата на кодирање, различни видови на тестирање, управување со промени. Практичните примери опфаќаат формална спецификација на научна апликација користејќи интервална темпорална логика, градење на научни работни текови користејќи ги алатките OpenMole и Taverna, примена на програмскиот модел Map-reduce и Hadoop во решавање на реални проблеми, дистрибуиран

развој на научни апликации, регресиско и агилно тестирање, како и управување со промени и контрола на верзии кај научни апликации.

Четвртото поглавје „Рамка за развој на научни апликации" опфаќа предлог модел за развој на научен софтвер каде што детално се опишуваат сите фази од развојот, модел за квалитет на научни апликации каде што се прави избор на атрибути и метрики соодветни за научните апликации. Поглавјето започнува со дефиниција на рамката за развој на научни апликации. Втората подглава „Модел за развој на научни апликации" ги прикажува постоечките модели на развој на софтвер кои ги нуди софтверското инженерство и содржи детален опис на сите фази на развој на софтверот. Во следната подглава се дефинира моделот за квалитет на научни апликации преку избор на атрибути и метрики кои се соодветни за научните апликации.

Во поглавјето „Примена на рамката за развој на научни апликации" се прави евалуација на предложената рамка за развој на научни апликации преку развој на апликација за решавање на еднодимензионална и дводимензионална Шредингерова равенка со користење на методот на претставување со дискретни променливи. За да се потврди значењето на примената на моделот за развој на научни апликации, користејќи го моделот за квалитет каде квантитативно преку метрики може да се пресметаат атрибутите кои влијаат на квалитетот на апликациите, се прави компаративна анализа на апликацијатата за решавање на еднодимензионална и дводимензионална Шредингерова равенка со користење на методот на претставување со дискретни променливи и на апликација за екстракција на бројот на срцеви отчукувања во минута (heart rate) и бројот на дишења во минута (respiratory rate) од ЕКГ сигнал (електрокардиограм). И двете апликации се напишани во програмскиот јазик C. На крајот се истакнуваат придобивките и се воочуваат можните недостатоци на рамката за развој.

Шестото поглавје е наменето за формулирање на заклучокот и идната работа. Користената библиографија е прикажана во последното поглавје.

1.6 Објавени трудови

1. Bojana Koteska, Ljupco Pejov, and Anastas Mishev. "Quantitative Measurement of Scientific Software Quality: Definition of a Novel Quality Model". In: International Journal of Software Engineering and Knowledge Engineering (2018), accepted, 2016 JCR Impact Factor 0.299.

2. Bojana Koteska, Anastas Mishev, and Ljupco Pejov. "Computational study of intramolecular O-H stretching vibrations in the two rotamers of free formic acid". In: Romanian Reports in Physics (2017), accepted, 2016 JCR Impact Factor 1.467.

3. Bojana Koteska, Anastas Mishev, and Ljupco Pejov. "Computational approach towards vibrational spectroscopic detection of molecular species relevant to atmospheric chemistry and climate science: The formic acid rotamers". In: IEEE EUROCON 2017-17th International Conference on Smart Technologies. IEEE. 2017, pp. 926–931.

4. Bojana Koteska, Anastas Mishev, Marija Glavas Dodov, Maja Simonoska Crcarevska, Jasmina Tonic Ribarska, Vesna Petrovska Jovanovska, Monika Stojanovska, and Ljupco Pejov. “Modeling the solid-state vibrational spectroscopic properties of morphine-based formulations with hybrid meta density functional theory”. In: IEEE EUROCON 2017-17th International Conference on Smart Technologies. IEEE. 2017, pp. 938–943.

5. Bojana Koteska, Goran Velinov, and Anastas Mishev. “A Novel Model for Measuring Component-Based System Agility”. In: Proceedings of the 8th Balkan Conference in Informatics. BCI ’17. Skopje, Macedonia: ACM, 2017, 5:1–5:4. ISBN: 978-1-4503-5285-7. DOI: 10.1145/3136273.3136295.

6. Bojana Koteska, Ljupco Pejov, and Anastas Mishev. “Pharmaceutical Software Quality Assurance System Architecture”. In: Proceedings of the 5th Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications, SQAMIA 2016. Vol. 1677. Budapest, Hungary: Faculty of Sciences, University of Novi Sad, Serbia, 2016, pp. 41–46. ISBN: 978-86-7031-365-1.

7. Bojana Koteska, Anastas Mishev, and Ljupco Pejov. “Isotropic Magnetic Shielding of Al(OH)-4 in Aqueous Solution: A Hybrid Monte Carlo - Quantum Mechanical Computational Model”. In: ICT Innovations 2015 : Emerging Technologies for Better Living. Ed. by Suzana Loshkovska and Saso Koceski. Cham: Springer International Publishing, 2016, pp. 39–48. ISBN: 978-3-319-25733-4. DOI: 10.1007/978-3-319-25733-4_5.

8. Bojana Koteska, Anastas Mishev, Ljupco Pejov, Maja Simonoska Crcarevska, Jasmina Tonic Ribarska, and Marija Glavas Dodov. “Computational Vibrational Spectroscopy of Hydrophilic Drug Irinotecan”. In: Proceedings of the Eighth International Conference on Advances in System Simulation - SIMUL 2016, pp. 11–16.

9. Verce Manevska, Bojana Koteska, Anastas Mishev, and Ljupco Pejov. “Quantum Vibrational Dynamics of Molecular Species Relevant to Atmospheric Chemistry and Climate Science. Formic Acid and its Clusters with Benzene”. In: XXIV Congress of Chemists and Technologists of Macedonia. Society of Chemists and Technologists of Macedonia. Ohrid, Macedonia, 2016.

10. Bojana Koteska, Anastas Mishev, and Ljupco Pejov. “Magnetic Response Properties of Aqueous Aluminum(III) Ion: A Hybrid Statistical Physics Quantum Mechanical Approach Implementing the Map-Reduce Computational Technique”. In: ICT Innovations 2014: World of Data. Ed. by Ana Madevska Bogdanova and Dejan Gjorgjevikj. Cham: Springer International Publishing, 2015, pp. 33–43. ISBN: 978-3-319-09879-1. DOI: 10.1007/978-3- 319-09879-1_4.

11. Bojana Koteska, Anastas Mishev, and Ljupco Pejov. “Two Approaches for Solving the Torsional Schrödinger Equation: Fourier Grid Hamiltonian Method

and Hamiltonian Diagonalization Method". In: Proceedings of the 12th Conference for Informatics and nformation Technology. Bitola, Macedonia: Faculty of Computer Science and Engineering, Skopje, Macedonia, 2015, pp. 216–220. ISBN: 978-608-4699-05-7.

12. Bojana Koteska, Ljupco Pejov, and Anastas Mishev. "Scientific Software Testing: A Practical Example". In: Proceedings of the 4th Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications, SQAMIA 2015. Vol. 1375. Maribor, Slovenia: Faculty of Sciences, University of Novi Sad, Serbia, 2015, pp. 27–34. ISBN: 978-961-248- 485-93.

13. Bojana Koteska, Anastas Mishev, and Ljupco Pejov. "Framework for Developing Scientific Applications: Solving 1D and 2D Schrödinger Equation by using Discrete Variable Repre- sentation Method". In: Proceedings of the First International Conference on Advances and Trends in Software Engineering - SOFTENG. 2015, pp. 93–99. ISBN: 978-1-61208-449-7.

14. Bojana Koteska, Anastas Mishev, and Ljupco Pejov. "A Robust Hybrid Statistical Physics- Quantum Chemical Approach to Magnetic Properties of Complex Aqueous Aluminimum (III) Species: Implementation on HPC Environment". In: Women in HPC, ISC High Performance. Frankfurt Am Main, Germany, 2015.

15. Bojana Koteska, Dragan Sahpaski, Anastas Mishev, and Ljupco Pejov. "On the Choice of Method for Solution of the Vibrational Schrödinger Equation in Hybrid Statistical Physics - Quantum Mechanical Modeling of Molecular Solvation". In: ICT Innovations 2013: ICT Innovations and Education. Ed. by Vladimir Trajkovik and Misev Anastas. Heidelberg: Springer International Publishing, 2014, pp. 187–196. ISBN: 978-3-319-01466-1. DOI: 10.1007/978-3-319-01466-1_18.

16. Bojana Koteska, Ljupco Pejov, and Anastas Mishev. "Formal Specification of Scientific Applications Using Interval Temporal Logic". In: Proceedings of the 3rd Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications, SQAMIA 2014. Vol. 1266. Lovran, Croatia: Faculty of Sciences, University of Novi Sad, Serbia, 2014, pp. 29–37. ISBN: 978-86-7031-374-3.

17. Bojana Koteska, Boro Jakimovski, and Anastas Mishev. "Building Scientific Workflows on the Grid: A Comparison between OpenMole and Taverna". In: RO-LCG 2014 Conference. Ma^gurele, Romania, 2014.

18. Bojana Koteska and Anastas Misev. "Change Management and Version Control of Scientific Applications". In: International Journal of Computer Science & Information Technology - IJCSIT 6.2 (2014), pp. 153–161. DOI: 10.5121/ijcsit.2014.6211.

19. Bojana Koteska, Ljupco Pejov, and Anastas Mishev. "Average Vibrational Potentials of Oscillators in Condensed-matter Environments using Hadoop". In: Proceedings of the 11th Conference for Informatics and Information Technology. Bitola, Macedonia: Faculty of Computer Science and Engineering, Skopje, Macedonia, 2014, pp. 311–314. ISBN: 978-608-4699-04-0.

20. Bojana Koteska and Anastas Mishev. “Software Engineering Practices and Principles to Increase Quality of Scientific Applications”. In: ICT Innovations 2012: Secure and Intelligent Systems. Ed. by Smile Markovski and Marjan Gusev. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 245–254. ISBN: 978-3-642-37169-1. DOI: 10.1007/978-3-642-37169- 1_24.

21. Bojana Koteska and Goran Velinov. “Component-Based Development: A Unified Model of Reusability Metrics”. In: ICT Innovations 2012: Secure and Intelligent Systems. Ed. by Smile Markovski and Marjan Gusev. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 335–344. ISBN: 978-3-642-37169-1. DOI: 10.1007/978-3-642-37169-1_33.

22. Bojana Koteska and Anastas Mishev. “A Software Engineering Perspective for Higher Quality Grid Distributed Development”. In: Proceedings of the 10th Conference for Informatics and Information Technology. Bitola, Macedonia: Faculty of Computer Science and Engineering, Skopje, Macedonia, 2013, pp. 247–251. ISBN: 978-608-4699-01-9.

23. Bojana Koteska, Ljupco Pejov, and Anastas Mishev. “Software Engineering Solutions for Improving the Regression Testing Methods in Scientific Applications Development”. In: Proceedings of the 2nd Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications, SQAMIA 2013. Vol. 1053. Novi Sad, Serbia: Faculty of Sciences, University of Novi Sad, Serbia, 2013, pp. 53–61. ISBN: 978-86-7031-269-2.

24. Bojana Koteska and Anastas Mishev. “Agile Software Testing Technologies in a Large Scale Project.” In: Local Proceedings of the Fifth Balkan Conference in Informatics, BCI 2012. Vol. 920. Novi Sad, Serbia: Faculty of Sciences, University of Novi Sad, Serbia, 2012, pp. 121–124. ISBN: 978-86-7031-200-5.

25. Bojana Koteska and Bisera Dugalic. “Recent and future trends and challenges of software testing”. In: Proceedings of the 9th Conference for Informatics and Information Technology. Bitola, Macedonia: Faculty of Computer Science and Engineering, Skopje, Macedonia, 2012, pp. 334–338. ISBN: 978-608-4699-01-9.

Глава 2

Научни апликации

2.1 Карактеристики на научните апликации

Секој физички објект со познати карактеристики може да биде моделиран и симулиран преку развој на научна апликација. Научните апликации се софтверски апликации кои симулираат активности и објекти од реалниот свет претворајќи ги во математички модели [222]. Научните апликации решаваат проблеми во различни научни области: пресметковна хемија и физика, биоинформатика, биологија, математика, итн.

Симулациите на научните проблеми најчесто немаат прецизни решенија и притоа се врши апроксимација со нумерички алгоритми. Научните апликации често решаваат комплексни проблеми кои не можат да бидат тестирани на стандарден начин бидејќи понекогаш мора да се изврши и физички експеримент или добиените резултати да се споредат со веќе докажани резултати. За да се покаже точноста на моделот, научниците треба да соберат информации од научните области и бази, каде што понекогаш информациите се ограничени [112]. Поради недостаток на научни бази, научниците се на некој начин принудени да судат врз основа нивното искуство за да ја проценат веродостојноста наместо точноста [90].

Верификацијата на научните апликации потврдува дека равенките се решени правилно без грешки во кодот и во најголем дел се поврзани со математичките модели. Валидацијата гарантира дека добиените резултати се прецизна репрезентација на проблемот и дека моделите за симулација се точни [164]. Доколку резултатите се точни, научниците може да ги прилагодуваат и теоријата и кодот. После тоа, потребно е да се подготват податоци и да се извршат тестови за да проверат излезите повторно [177]. Процесот на откривање на грешки може да се подобри со користење на модели за предвидување на грешки кои се базираат на математички модели и грешки кои се случиле во претходните верзии на апликацијата.

Извршувањето на комплексните математички формули и симулации, вообичаено бара суперкомпјутер и пресметување со високи перформанси. Во научните апликации спаѓаат и апликации кои се користат за вградување во инструменти, за обработка, анализа, или визуелизација на податоци, како и за организирање на работни процеси [183]. Исто така, постојат научни апликации кои се дистрибуирани географски и различни групи работат на апликацијата.

Притоа, често постои и ограничено познавање на дел од софтверските компоненти [34] со што уште повеќе се отежнува тестирањето и понатамошниот развој на апликацијата.

2.2 Категоризација на научните апликации

Категоризацијата на научните апликации е важна за одредување на можните разлики во процесот на развој и таа може да се направи со воочување на различните карактеристики на софтверот. На пример, научните апликации може да се категоризираат според: големина (линии код - LOC), цел, извршувачки тип, број на корисници, тип на корисници (блискост со доменот). Можна поделба според дадените карактеристики е прикажана во Табела 2.1 [177].

Табела 2.1 – Категоризација на научни апликации

Големина (LOC)	Мала (<1,000)	Средна (≥ 1,000, <100,000)	Голема (≥ 100,000)	
Цел	Комерцијална	Истражувачка	Образовна	
Тип на извршување	Сериски		Интерактивен	
Број на корисници	Еден	5-10	10-100	>1000
Тип на корисници	Ист домен	Научници кои не се специјалисти	Научници од други домени	Други професии

Категоризацијата на научните апликации е значајна при процесот на проверка на резултатите. Некои од резултатите се проверуваат со споредување со резултатите добиени од физички експерименти. Од друга страна, ако тоа не е можно, резултатите се проверуваат со споредување со веќе постоечки резултати или пак се базираат на теорија. Со цел да се изврши проверка на резултатите правилно, следните работи треба да бидат земени во предвид: доменот на апликацијата, статистички податоци, експериментални резултати и метрики за евалуација. [154].

Според [91], научните апликации се составени од: основна компјутерска инфраструктура (на пр. оперативни системи), работна групи, софтвер-посредник и надворешни компоненти (обично со отворен код). Исто така, компонентите може да се карактеризираат по честота на нивна употреба и нивната употреба во различни научни контексти.

Програмскиот јазик е исто така важен дел при поделба на научните апликации. Некои програмски јазици се дизајнирани за нумерички пресметки и тие брзи, тешки за учење и не комуницираат лесно со графички или алгебарски процеси. Од друга страна, постојат стандардни научни пакети кои се добри за интеграција на нумерика, алгебра и графика, но се бавни и ограничени во даден опсег [195].

2.3 Споредба на научни апликации и комерцијални апликации

Научните апликации вообичаено се користат во истражувачката група во која се развиваат и најважната работа е да се добие научно точен резултат со добри модели за симулација доставени навреме [25, 181], додека комерцијалните апликации се дизајнирани и развиени за продажба на клиенти [161].

Верификацијата и валидацијата на научните апликации не може да биде изведена на ист начин како кај комерцијалните апликации, бидејќи процесот на дефинирање на барањата кај научните апликации е тежок и не постојат надворешни корисници. Верификацијата на комерцијалните апликации е процес кој гарантира дека овој софтверски производ е во согласност со одредени барања, ги задоволува стандардите и конвенциите, додека верификацијата кај научните апликации гарантира дека равенките се решени правилно и соодветствуваат со дефинициите на математичкиот модел. Од друга страна, валидацијата на комерцијалните апликациите проверува дали софтверот ги задоволува барањата на корисниците. Валидацијата кај научните апликации потврдува дека резултатите добиени од кодот се точна репродукција на проблемот, а тоа се потврдува со споредување на резултатите со експериментално добиените податоци [164].

Според интервјуто спроведено со шеснаесет научници и инженери прикажано во [176], авторите откриле дека валидацијата на научниот софтвер се базира на професионално расудување (пр. визуелна споредба помеѓу две слики), споредба на резултатите од моделот со други модели и сл.

Генерирањето на тест случаи е исто така различно кај комерцијалните и научните апликации. Тестирањето на научни апликации се потпира главно на моделите за тестирање кои претставуваат симулации на проблеми во реалниот свет. Тестовите кај научните апликации се повеќе фокусирани на правилна употреба на математички библиотеки и нивните имплементации, а тестовите кај комерцијалните апликации се поврзани со одредени барања [27]. Научните апликации не може да се оценуваат од страна на клиентите, бидејќи тие се развиени за решавање на конкретни проблеми во научната заедница, а не за деловни цели.

При развој на научен софтвер, научниците се принудени на некој начин да користат строго одредени програмски јазици, а тоа претставува дополнителен проблем. Употребата на нови технологии кои можат да го олеснат овој процес често побарува дополнителни ресурси или време за интеграција со постоечкиот код. Сите овие фактори влијаат на продуктивноста на некој начин. Во овој контекст, главните недостатоци се: недостаток на јазици специфични за доменот, недостаток на флексибилна и ефикасна инфраструктура која треба да обезбеди соодветни брзи промени и неправилно разбирање на моменталната состојба на системот [85].

2.4 Научниците и научните апликации

Сегашниот развој на научните апликации започнува со фазата на кодирање. Сегал и Морис го опишуваат развојот на научни апликации како процес кој започнува со идеја, продолжува со пишување дел од кодот и доколку не се потребни промени, процесот завршува [183].

Научниците не практикуваат пишување барања или било каков вид на документи и тие веруваат дека за барањата треба да се разговара, а не да бидат запишани [181]. Фазата на спецификација на барања не е официјално документирана. Еден аспект на проблемот за спецификација на барањата е тоа што софтверските инженери не можат да ја разберат теоријата, а научниците немаат формално образование од областа на софтверско инженерство. Иако целта на спецификацијата на барањата на софтверот е да помогне да се сфати проблемот, раната спецификација на софтверските барања при развој на научен софтвер е понекогаш тешка или невозможна [80].

Научниците вообичаено учат да програмираат сами или други научници им пренесуваат искуства. Тие сметаат дека процесот на развој на софтвер е само на процесот на кодирање. Апликациите кои ги развиваат се наменети за нивната научна група или поблиската научна заедница. Најважната работа за научниците е да се добијат научно точни резултати [25, 181]. Ако резултатите на излез се точни, тие не се многу заинтересирани за дополнителни оптимизации или паралелизации на кодот. Тие често се водат од мислата дека ако хипотезата е докажана еднаш, нема потреба за повторно програмирање на тој дел [27].

Научниците пишуваат кодови во мали тимови без претходна формална обука [174]. Причината за слабиот квалитет на апликациите лежи во недостатокот на формални методи и практики на софтверското инженерство [25, 182, 209]. Тоа може да резултира со поголем број на грешки, тешко разбирање на кодот напишан од страна на други научници во заедницата, користење на дополнителни непотребни ресурси, проблеми со рефакторирање и оптимизации подоцна, итн. [181].

Кога се врши тестирањето, нема информации за тоа што е направено и што треба да се направи. Кога ќе започне процесот на верификација, научниците даваат значење на квалитетот на алгоритмот, а не на неговата реализација [25]. Вообичаено, научните проблеми немаат прецизно решение и се користат нумерички методи за апроксимација. Често, проверката значи споредба со експериментални резултати [174]. Верификацијата на софтвер се базира на професионално расудување, како што е споредба на излезот од моделот со излезите од друг модел (benchmark). Исто така, се користи и визуелна споредба помеѓу слики [176]. Примарна задача на научниците е да се тестира теоријата, а не имплементацијата на алгоритмот. Тестирањето не е најважното нешто, сè додека не се појави грешка која влијае на точноста на научните резултати [182].

Според одговорите дадени во [122], научниците се свесни за важноста на процесот на тестирањето, но вистинското прашање е како да се направи тестирање. Вообичаено, програмерите се единствените одговорни лица за тестирањето и тие не се сигурни дали во барањата постојат грешки. Научниците

преферираат тестирање на бела кутија (white-box testing) и рачно испитување на кодот, наместо примена на автоматско тестирање и алатки за тестирање. Тестирањето се врши после процесот на развој на софтверот. Според истражувањето, научниците применуваат функционално и интеграциско тестирање, но, тоа го прават без план и документација. Практиките за користење на регресиско тестирање и тестирање по опоравување од грешки се речиси незначителни.

Тест случаите се опишани слободно без никаква употреба на стандардизирани форми. Тие најчесто се базираат на модели и анализи на изворниот код, а не се користи специфична техника за генерирање на тестови како што е анализа на гранични вредности или поделба по категории. Научниците не применуваат техники за дизајнирање на тестови базирани на структура, како што се покривање на наредби, покривање на гранки, покривање на услови и на патишта во графови за изворниот код. Научниците веруваат дека тестирањето е завршено кога сите тест случаи ќе се извршат без грешки. Пронајдените грешки вообичаено ги коригираат според нивниот приоритет.

Научниците кои имаат искуство во развој на слични апликации ги користат истите техники за генерирање на тест случаи во други апликации. Резултатите покажуваат дека прелиминарните испитувања и погодувања на грешки се многу важни. Времето претставува најголема бариера за развој, бидејќи најчесто тимовите имаат ограничено време да ги користат ресурсите, а повеќето од нив сметаат дека повеќе од важно е да се направи значителен напредок во истражувањето, отколку во имплементацијата на тестирањето.

Учесниците во анкетата не применуваат техники за следење на барањата или верзиите на софтверот. Истражувањето покажа дека развојот на научни апликации резултира со чести промени во барањата кои зависат од научниот напредок во решавањето на проблемот. Доколку се појави грешка во новата верзија на софтверот, ќе биде полесно да се најде, доколку постои информација за извршените измени. Тимовите кои развиваат научни апликации не креираат документи од процесот на тестирањето, како што се план за тестирање, скрипти за тестирање, тест случаи, логови за грешки и сл.

2.5 Софтверското инженерство во развојот на научните апликации

Достигнувањата во областа на природните науки во последниве години, како и постојаниот технолошки напредок поставува отворено прашање за тоа како софтверското инженерство може да биде активно вклучено во процесот на создавање на научни апликации и како може значително да го подобри квалитетот на нивниот развој.

Софтверските апликации создадени да симулираат научни феномени се од клучно значење за науката и се особено корисни кога станува збор за автоматизација на одредени делови од процесот на развој на апликациите, како што се управување со работни процеси, активности при тестирање, анализа на податоци, визуелизација, итн. Создавањето на научните апликации не само што придонесува за напредокот на одредена научна област, но, исто

така, создава можности за полесна соработка и размена на искуства, идеи и иновации помеѓу научниците [92].

Една од најважните работи во сегашниот процес на развој на научни апликации е да се подобри квалитетот со цел да се создаде поефикасен процес каде што ќе се откријат и елиминираат грешките во најкраток можен рок.

Според стандардтот IEEE Std 610,12-1.990, софтверското инженерство се дефинира како примена на систематски, дисциплиниран и квантитативен пристап кон развојот, функционирањето и одржувањето на софтвер [3]. Софтверското инженерство, исто така, преставува формално множество на процедури и алатки кои треба да обезбедат ефикасен процес на развој [174].

Проблемот со развојот на научните апликации е практиката на развој. Процесот на развој на научни апликации не ги следи правилата за развој кои ги предлага софтверското инженерство, но и развојот на научните апликации се разликува од развојот на комерцијалните апликации, што значи дека истите практики за развој не може да се применат во целост, туку треба да се направат соодветни прилагодувања.

Имајќи в предвид дека главната разлика помеѓу развојот на комерцијалните и научните апликации е комплексноста на доменот [182], процесот на одредувањето на квалитетот на софтверот е тешка задача [72] и вистински предизвик за софтверските инженери е да се најде соодветна комбинација, адаптација и модификација на постоечките практики кои ги нуди софтверското инженерство. Интеграцијата на практиките на софтверското инженерство во процесот на развој на научните апликации треба да се направи внимателно, бидејќи неправилната комбинација може да предизвика негативен ефект, што е спротивно на очекувањата.

Се тврди дека апликациите кои користат пресметување со високи перформанси, како што се научните апликации, во основа можат да бидат исти со било каков тип на апликации, но потребни се мал број на дополнителни промени во однос на планирањето, откривањето на барањата, тестирањето и пристапот за развој [27]. Развојот на големи и сложени научни пресметковни симулации бара добро управување со проектот, идентификација на ризици, инженерство за квалитет на софтверот, методи на валидација и верификација [164]. Според Басили, развојот на научните апликации се состои од два чекора: прво, проблемот треба да биде преведен во алгоритам и второ, алгортимот треба да се преведе во код [25].

Софтверското инженерство нуди алатки и методи кои можат да помогнат во процесот на дефинирање на барањата, пишувањето на кодот, тестирањето, управувањето на софтверот, итн. Со оглед на тоа дека првично софтверското инженерство е создадено за научни и комплексни системи [174], во голема мера може да влијае на квалитетот и продуктивноста на научниот софтвер на два начина: со воведување на нови инженерски практики кои се покажале дека се добри во големи проекти, како и примена на тие кои се веќе адаптирани [25]. Дел од практиките на софтверското инженерство кои треба да се земат в предвид се: [164]: план за развој на задачи и одговорности, управување со барања, креирање на документација, идентификација на ресурси, управување со ризици, управување со промени и постојана контрола на квалитетот.

Со оглед на тоа што големите софтверски системи се обично напишани во неколку различни програмски јазици, дизајнот на архитектурата, како и коректниот избор на програмски јазик за развој на научните апликации може значително да придонесе за подобра организација на развојот и справување со можните грешки.

Постојат повеќе трудови во кои се потенцира дека примената на софтверското инженерство може да го подобри квалитетот на научните апликации. Истражувањата покажуваат дека некои практики се успешно вклучени во развојот на научните апликации, но сè уште нема докази дека тие може да се користат како такви. Со големиот пораст на интересот за создавање научни апликации и благодарение на брзиот технолошки напредок и поддршката што се нуди за овој тип на апликации, мотивот за понатамошни истражувања во оваа област станува уште поголем.

Хуанг и Лап дискутираат за причините зошто вклучувањето на софтверското инженерство во процесот на развој на научни апликации е вистински предизвик [93]. Тие тврдат дека вклучувањето на софтверското инженерство може да им помогне на научниците лесно и транспарентно да инвестираат во подобра иднината на истражувачката заедница. Во својот труд [174], Роj дава преглед на современите практики на софтверското инженерство, особено оние кои се релевантни за развојот на научни апликации. Тој опишува модели за развој на софтвер, избор на јазик за програмирање, системи за контрола на верзиите на софтверот, алатки за статичка анализа на код, процедури за динамичко тестирање на софтверот и квантитативна проценка на квалитетот и сигурноста на софтверот. Во [16], се нагласува потребата од дефинирање на кориснички случаи, шаблони за дизајн, системска архитектура и агилни принципи на развој како важни методи во развојот на научен софтвер.

Во [83], авторите ги презентираат резултатите од истражувањето каде се анализира моменталната состојба на користење на практики на софтверското инженерство во областа на компјутерски науки и развој на софтвер. Тие откриваат неколку проблеми при развој на софтверот и предлагаат решенија на овие проблеми. Во [142], авторите презентираат интердисциплинарен истражувачки проект чија цел е да се развијат нови методи на софтверското инженерство за дистрибуирано и паралелно научно пресметување. Нивниот проект го евалуираат во пракса и даваат извештај за напредокот. Во [210], авторите препорачуваат 25 практики за научен развој на софтвер кои можат да ја подобрат продуктивноста на научниците и квалитетот на научните апликации.

Едни од практиките за развој на научни апликации развој кои ги нуди софтверското инженерство се следниве [164]:

- Управување со барања, нивно документирање и постојана контрола;
- Развивање на план и распоред за завршување на задачите и одговорностите;
- Идентификација на ресурсите потребни за развој на апликацијата;
- Управување со ризиците кои можат да произлезат во текот на развојот;

- План за тестирање и соодветна документација;
- Управување со промени;
- Контрола на квалитетот.

Една од алатките која го поддржува процесот на развој на научни апликации е алатката ViPER. Алатката е базирана на Eclipse, нуди визуелен интерфејс и го поддржува инженерството базирано на модели, UML моделирање, ANSI-C генерирање на код, како и управување со кориснички барања. ViPER овозможува и планирање на развој, валидирање на изворните артефакти и регресиско тестирање [89].

2.6 Спецификација на софтверски барања

Една од најважните работи при изработката на една научна апликацијата е концизна спецификација на барањата во почетната фаза на софтверскиот развој. Процесот на спецификација на барања кај научните апликации не може да се спроведе во целост на стандарден начин како кај комерцијалните апликации. Тоа се должи на честите промени кои се случуваат постојано [160, 16]. Исто така, при развој на научни апликации сите барања не можат да се предвидат однапред [174] и треба да бидат дефинирани во итерации [102].

Барањата може да се поделат на два типа: барања од корисниците и барања за софтверскиот систем. Корисничките барања се напишани во форма разбирлива за корисниците, а софтверските барања претставуваат формална дефиниција на функциите и ограничувањата на софтверскиот систем. Софтверските барања може да се поделат во три категории: функционални, нефункционални и барања од доменот [174].

Во [140], авторите предлагаат модел за специфични барања од доменот кој вклучува научни сознанија како клучен фактор за идентификување на барањата. Барањата треба да бидат специфицирани користејќи стандардни шаблони [2, 5, 6]. Правилно специфицираните барања се корисни во фазата на креирање на тест случаи.

2.7 Моделирање на домен и пишување на код

Моделирањето на доменот, генеричкото програмирање и програмирањето базирано на компоненти може исто така да го подобрат квалитетот на научните апликации. Создавањето на мета модел, спецификација на концептите и нивната организација во процесот на моделирање на доменот помагаат при дефинирање на моделот на апликацијата, вклучувајќи познати концепти за решавање на проблеми специфични за тој домен. Квалитетот се подобрува со тоа што се воспоставуваат одредени правила во мета моделот, а бројот на грешки е помал бидејќи кодот кој веќе е проверен од страна на експерти во конкретната област се создава со користење на алатки [87].

Во [45], авторите презентираат модел за развој на научен софтвер кој ги спојува софтверските инженери и научниците и ги вклучува принципите на

агилниот развој, како што се итеративен и инкрементален развоен циклус, и пристап за развој на софтвер со отворен код. Овој пристап се состои од четири фази: дизајн, развој, подобрување и објавување. Во [99], авторите предлагаат рамка за развој на софтвер прилагодлива на сите фази на развој на софтвер која вклучува барања за квалитет, контрола на квалитет и воспоставување на квалитет на софтверот.

Со оглед на фактот дека често се случува големите софтверски системи да бидат напишани со користење на неколку различни програмски јазици [72], дизајнот на архитектурата, како и правилниот избор на програмски јазик може значително да придонесе за подобра организација на кодот и справување со грешки [174].

Проблемите поврзани со програмскиот код, во минатото се решавале така што кодот се препишувал уште еднаш врз основа на стариот код. Во денешно време кога има многу посложени и поголеми кодови, најдобро решение е да се направи рефакторирање на кодот [153]. Рефакторирањето на кодот кај научните апликации има големо влијание врз намалување на комплексноста на кодот, времето потребно за извршување на пресметки и меморијата. Сепак, процесот на рефакторирање бара постојано тестирање, бидејќи не треба да предизвика промена на резултатот, туку само во алгоритмот [174].

Паралелизацијата на кодот, исто така, е многу корисна практика во процесот на развивање на научен софтвер. При паралелизација може да се вклучат неколку практики на софтверското инженерство, како што се: објектно - ориентирани нотации, генеративно програмирање, аспектно ориентиран софтверски развој, итн. Овие практики помагаат во процесот на трансформирање на нотациите, со цел да се задоволат софтверските барањата и да се постигне оптимизација. Со цел да се постигне успешна паралелизација на кодот, проблемот мора да се подели на помали логички единици. Поделбата може да се реализира со креирање на граф и истото може да се направи со алатки кои го поддржуваат овој процес [189]. Процесот на паралелизација обично значи и оптимизација на кодот [143].

Постигнување на подобра прегледност и помала можност за грешки во процесот на програмирање може да се постигне на следниот начин [174]: користење на долги описни имиња за променливи, функции, објеки, итн.; пишување на коментари во кодот; енкапсулација на податоци; справување со исклучоци како што е делење со нула, недостаток на меморија, недостаток на влезна датотека, итн.; соодветно вовлекување на редовите при кодирање и сл.

Повторното искористувањето на веќе постоечки компоненти или алгоритми во процесот на развој на масивни системи го намалува времето потребно за програмирање. Потребно е да се процени дали навистина е потребно повторно искористување на код и дали тоа ќе придонесе за реализација за потребната функционалност [221]. Во ваков случај, треба да се спроведат тестови за да се утврди дали повторното искористување на код води во вистинската насока на добивање на точни резултати. Софтверските инженери можат да им помогнат на научниците да ги воочат предностите на повторното искористување на код преку употреба на рамки. Рамките може да го намалат напорот и времето потребно за развој, особено кога научниците веќе имаат знаења за технологиите кои ќе ги користат (на пример стандардот Message Passing Interface

(MPI)). Научниците сметаат дека учењето на стандардот MPI не е тешко, но ефективното програмирање со MPI е тежок процес. Затоа, организацијата на специјализирани курсеви, каде што се посветува особено внимание на квалитетот на апликациите ќе им овозможи на софтверските инженери да ги споделат своите знаења со научниците [25].

Повторното искористување на код е многу корисно, особено кога апликациите припаѓаат на иста научна област. Постојат многу различни пристапи и техники за повторно искористување на код, како што се [131]: софтверски рамки, библиотеки, генератори на код, шеми за дизајн, итн. Користењето на компонентно - базирана архитектура за развој на научни апликации може да ја подобри продуктивноста и перформансите на системот што е особено важно ако постои ограничен период за развој на системот. Денес, постојат голем број на достапни компоненти за научно пресметување кои нудат различни алгоритми, визуелизација, паралелно програмирање, влезови и излези, оптимизација, решавање на линеарни и нелинеарни равенки, итн. [13].

При паралелизација и оптимизација на кодот треба да се внимава бидејќи може да настанат промени во логиката на кодот и редоследот на извршување на наредбите. Тестирање во овој случај може да се спроведува со користење на теорија базирана на логика и алатки кои го поддржуваат тоа. Овој метод е прилично скап и бара големо познавање на математичката логика [21]. Интел има развиено технологија за автоматско дебагирање и детекција на грешки во апликациите кои го користат програмски модел MPI (Message Passing Interface) за постигнување паралелизам. Алатката нуди графички интерфејс и детектира грешки што може да се случат за време на пренос на пораките, како и грешки кои водат до ќорсокак и проблеми со ресурсите кои се неопходни за извршување на апликацијата [60].

Проектот за креирање на софтверска инфраструктура за научни пресметки (ACTS) го помага развојот на научните апликации преку голем број на бесплатни алатки. Алатките се поделени во четири категории: алатки за нумерички пресметки, развој на код, извршување на кодот и развојни библиотеки. Овие алатки обезбедуваат поддршка при оптимизација, добивање на резултатите од анализа, процесот на визуелизација, итн [146].

Дебагирањето кое помага во процесот на откривање на грешки е поддржано од неколку алатки и шаблони. Примери на вакви алатки се TotalView и gdb. Може да помогнат и доколку се применуваат кога се користат повеќе процесори за извршување на кодот [88].

2.8 Тестирање на научни апликации

Валидацијата и верификацијата на научните апликации не може да се дефинира на ист начин како кај комерцијалните апликации, бидејќи не постојат надворешни корисници кои може да се вклучат во процесот на спецификација на барањата. Верификацијата на софтверот е поврзана со математичкиот модел на системот и гарантира дека равенките се решени правилно, без грешки во кодот. Валидацијата гарантира дека резултатите добиени од изворниот код ќе бидат пандан на проблемот од природата, односно моделите кои се потребни за да се симулираат феномените на природата изразени преку код ќе бидат

точни. Валидацијата се врши преку споредување на резултатите со добиените податоци од извршените експерименти [164].

Суштинската разлика при тестирањето на научните и комерцијалните апликации е во дизајнот на тестови. Тестовите за научните апликации се насочени кон нумеричка точност и правилна употреба на математички библиотеки, додека концептите на регресиско и автоматско тестирање остануваат исти како кај комерцијалниот софтвер [27].

Тестирањето на научните апликации најчесто се реализира преку модели кои претставуваат симулација на проблемите од реалниот свет. Потребно е да се направат повеќе тестови за да се подобри квалитетот на научните апликации и да се провери функционалноста на моделот. Најтешката задача во овој процес е дефиницијата на тест случаи бидејќи може да се случи крајниот резултат на тестот да не е познат од почеток. Во овој случај потребно е да се направи колекција на проблеми од научната заедница, односно да се создадат бенчмарк тестови со анализирани решенија кои ќе помогнат во одредување на крајните резултати [221, 27].

Систематски преглед на факторите за успех и бариерите во процесот на подобрување на квалитетот на софтвер се прикажани во [113]. Алмас и останатите автори даваат предлог за ефикасен, автоматизиран работен тек за валидација на модели во хемиските процеси од пресметковна квантна хемија кој се мери со пресметковната брзина и неговата способност да издестилира големи податочни множества во робустни, концизни и информативни метрики [14]. Кели и Сандерс ги препорачуваат следните пристапи за оценување на научен софтвер дефинирани како дел од заедницата за софтверско инженерство: динамичко тестирање, статички анализи на код и формални методи [112].

Генерирањето на тест случаи може да се прави на неколку начини: генерирање на тест случаи врз основа на документи креирани за спецификација на кодот, врз основа на резултати добиени од анализа на граничните вредности, создавање на тест случаи врз основа на претпоставки за грешки и тестирање на 90% од наредбите во кодот [42]. Верификацијата и тестирањето на резултатите ќе се подобрат доколку апликацијата е достапна на увид и на други научници во оваа област кои би можеле да учествуваат во процесот на поправање на евентуални грешки [16].

Тестирањето на научните апликации каде што конечниот резултат не е познат однапред може да направи со создавање на симетрични тестови. Примери за такви тестови се транслациона и ротациона симетрија на физички системи (временска симетрија). При вакво тестирање, треба да се посвети посебно внимание на процесот на заокружувањето грешки, особено во хаотични системи [35].

Доколку апликацијата има големо множество на влезни податоци, најдобро решение за изработка на тестови е моделирање на влезниот домен, независните и зависните параметри. Истото може да се направи со помош на граф, каде темињата претставуваат параметри, а влезните рабови на темето се вредности за параметрите. Еден пат во графикот претставува еден тест случај [205].

При тестирањето на научните апликации може да се направи поделба на грешките во однос на тоа дали тие се дел од софтверот или во математичкиот модел [80]. Дополнително, предвидувачките модели базирани на математички

модели и статистичките податоци за бројот и видовите на грешките кои се случиле во претходните верзии, може да се искористат за предвидување на грешки кои може да се појават во некоја идна верзија на софтверот [162]. Според истражувањата, 40% од причините за пад на софтверот може да се отстранат со статичка анализа на код. Освен статичко, може да се примени и динамичко тестирање или тестирање за време на извршувањето на програмата [174].

Во [201], авторите ја истражуваат теоријата на дистрибуција на грешки, користејќи ја квантитативната анализа направена од страна на Фентон и Олсон за верификација на делови од големи софтверски системи. Тие спроведуваат студија за четири развојни проекти на последователни изданија од истиот комплексен софтверски производ за телекомуникациска размена.

Регресиското тестирање кое претставува повторно извршување на тест случаите со цел да потврди дека системот функционира правилно и после направените модификации [44] е тема на истражување во повеќе трудови. Во [188], авторите опишуваат некои техники за регресиско тестирање: селекција, приоритетизирање на тестови и хибридни техники. Техниката на селекција избира одредени тестови од старите тест случаи и ги извршува на новата изменета верзија на софтверската апликација. Техниката на приоритетизација го менува редоследот на извршување на тест случаите со цел да се подобри ефикасноста на тестирањето. Хибридните техники ги комбинираат и двете претходни техники. Пристап за приоритетизирање на тест случаи кои треба да се вклучат во процесот на регресиско тестирање е претставен во [192]. Приоритет на тест случај се дава врз основа на четири фактори: менување на специфицираните барања на софтверот, приоритет на клиенти, комплексност на изведување на тестовите, како и склоност кон грешки во барањата. Овој пристап не е целосно применлив во развојот на научен софтвер, бидејќи вистински клиенти не постојат. Факторот „Приоритет според клиентите" мора да се отстрани. Алгоритам за приоритет на тест случаи е опишан во [193]. Овој алгоритам дава приоритет на тест случаи со цел да ги максимизира грешките кои може да се случат за време на извршување на изворниот код. Добар пристап е да се користи хибридната техника дефинирана во [214] која претставува комбинација на модификација, минимизирање и селекција со приоритетизирање користејќи листа на промени на изворниот код и историја на извршувањето на тест случаи од претходните верзии на софтверот.

Во [169], авторите предлагаат околина за регресиско тестирање, инфраструктура за тестирање и системот DUNE (Дистрибуирана и унифицирана нумеричка околина). DUNE е слободна софтверска рамка за решавање на парцијални диференцијални равенки со методи базирани на мрежи [26]. Се создаваат тестови кои поддржуваат верификација и валидација на научни алгоритми. Во [112], авторите го гледаат регресиското тестирање како фактор за подобрување на квалитетот на развојот на научен софтвер. Совети за претворање на грешките во тест случаи се дадени во [209]. Лошите практики на научниците при регресиско тестирање се прикажани во [112]. Според оваа студија, научниците вообичаено не вршат систематско регресиско тестирање, туку применуваат ад-хок тестирање. Исто така, тие ги користат истите влезни податоци за регресиско тестирање повеќе од една година.

Регресиските тестови кои се состојат од споредба на резултатите од симулација со референти резултати не се практични [170]. Тие се користат за потврда на експериментални резултати, но реба да се избегнуваат бидејќи разликите во резултатите можат да бидат предизвикани од подобрувањето на кодот. Според резултатите од истражувањето прикажани во [84], регресиското тестирање е важно за научниците, но тие не го применуваат правилно бидејќи не се доволно запознаени со тоа. Исто така, развивачите на софтвер не применуваат автоматско регресиско тестирање. Од друга страна, важно за ова истражување е врската помеѓу промените на барањата и изборот на тестови за регресиско тестирање. Пристап кон регресиско тестирање кое користи софтверски барања и ги поврзува со тест случаите наместо со изворниот код е претставен во [43].

Едни од главните причини за примена регресиско тестирање се следниве: додавање или менување на функционалноста и откривање на грешки, поправање на грешки, додавање или адаптирање на постоечките функционалности или пренесување на софтверот во различни средини [31]. Друга причина за регресиско тестирање е рефакторирањето. Во [167], авторите го покажуваат односот помеѓу рефакторирањето и регресиското тестирање притоа правејќи анализа за влијанието на промените.

Автоматското тестирање на кодот се врши со алатките за анализа на код кои нудат можност за дефинирање на тест случаи кои ги покриваат наредбите од кодот кои сè уште не се покриени. Вршењето анализа за покриеност на кодот преку тест случаи помага за откривање на грешките при дизајн и имплементација [221]. Една од алатките која овозможува следење на грешки е Bugzilla [51].

Креирањето на автоматизирана рамка за тестирање е корисна за анализа на кодот и управување со пронајдените грешки. Рамката треба да обезбедува поддршка за тестирање на модули, регресиско тестирање, како и тестирање на целиот систем. Исто така, добро е да овозможи конзистентно изминување и организација на дефинираните тест случаи, како и приказ на статистички резултати добиени од процесот на тестирање на апликацијата [221].

Софтверските алатки за евалуација на големи системи го подобруваат квалитетот на софтверот преку подобрување на стабилноста и коректноста на имплементираните алгоритми. Постојат повеќе алатки со едноставни интерфејси кои овозможуваат поголема интероперабилност што е многу важно за научните апликации каде промените се случуваат многу често [146, 72].

2.9 Управување со промени и контрола на верзиите на софтверот

Со оглед на тоа дека развојот на научни апликации содржи постојани промени кои се резултат од корекција на грешки и додавање на нови функционалности [204], контролата на верзиите на софтверот е значајна за документирање на промените во софтверот и обезбедува можност за споредба на резултатите добиени од различни верзии на софтверот.

Управувањето со промени на софтверот поддржува менување на софтверот, притоа обезбедувајќи повторно функционирање и одржување на софтверот [77]. Целта на управувањето е да обезбеди постојана конзистентност на системот [145]. Потребата за управување со промените настанува од постојаните промени во кодот кои можат да предизвикаат дополнителни промени [139].

За успешно справување со промените, корисно е да се создаде прелиминарен план за управување со промени врз основа на претходни искуства и информации за видовите на промени кои се случиле при развој на софтвер кој решава сличен проблем. Главната корист на планот за управување со промени е можноста софтверот да се врати на претходната стабилна верзија, кога ќе се случи непредвиден пад или грешка кај софтверот.

Планот за управување со промени содржи информации за процесот на управување со промените и тоа е дел од планот за управување со проектот [194]. Тој обезбедува централнен информациски систем за софтверските промени. Постојат неколку модели за управување со промени кои се наменети првенствено за развој на комерцијален софтвер. Во [73], авторите предлагаат спирален модел за развој на софтвер составен од 4 циклуси: спецификација на ново барање или проблем со постоечките производи, решавање на проблеми од нетехничка гледна точка, примена на системско инженерство и технички решенија.

Управувањето со промените во софтверот претставува важен процес во фазите на кодирање и тестирање на софтверот кога се користат системи за следење на грешки [133]. Управувањето со промени и следењето на промени се корисни и за процесот за управување со ризици [65].

Контролата на верзиите на софтверот е процес на идентификување и следење на различни верзии на софтверот. Таа нуди управување со изворниот код, електронски документи, хартиени документи и други артефакти [12]. Одговорноста на системот за контрола на верзии е решавање на конфликтите кои се јавуваат како резултат на истовремени промени на исти делови од кодот. Овој систем треба да обезбеди согласност за спојување на изменетите делови. Исто така, системот е корисен кога треба да се направат некои подобрувања на кодот (оптимизација на изворниот код, промена на редоследот на наредбите и паралелизација на кодот).

Според резултатите од спроведеното испитување [175], постојат испитаници кои воопшто не вршат контрола на верзиите на научен софтвер, а некои од нив го прават тоа без користење на алатки. Мал дел од испитниците постојано користат соодветни алатки. Ист таков пример за недостаток на знаење од страна на научниците за системи за контрола на верзиите на софтвер се прикажани во [211].

Придобивките од управувањето со промени и користење на системите за контрола на верзиите се истакнуваат во [89]. Како придодобивки се набројуваат: намалување на губитокот на информации, зголемување на актуелноста на документите и следење на влијанието на промените. Стариот начин на проследување кодови преку електронска пошта или со користење на Dropbox, треба да се замени со употреба на софтверски системи за контрола на верзиите [209].

Системите наменети за вршење на контрола на верзиите на софтверот обезбедуваат можност повеќе корисници да го развиваат целиот изворен код паралелно и да го синхронизираат подоцна. Добиените податоци од истражувањето спроведено во [122] го потврдуваат некористењето на алатки за контрола на верзиите на софтверот и управувањето со промени. Како последица на тоа, голем број тимовите кои развиваат научен софтвер не сигурни дали нивниот софтвер содржи грешки или не. Лошите практики за развој на научен софтвер, исто така, се потврдуваат и преку резултатите добиени од две други истражувања [80, 25].

Глава 3

Техники на моделирање и развој на научни апликации

3.1 Формална спецификација на научни апликации

3.1.1 Досегашни истражувања

Постојат одредени истражувања поврзани со формална спецификација на софтверски системи и архитектури. Во [157] авторите даваат преглед на тоа како формалните методи можат да се применат во развојниот процес на еден софтверски систем. За проблемите поврзани со верификација со конечни состојби на научни апликации кои користат MPI се зборува во [186]. Се предлагаат и некои теореми за поедноставување на моделите со конечни состојби кај програмите кои користат MPI. Во [187], авторите применуваат техники на проверка на моделот на софтверот BlobFlow кој претставува научен софтвер составен од десетици илјади линии код и е наменет за решавање на дводимензионални равенки на Navier-Stokes. Интерактивен систем за изведување на формална спецификација на програми и податочни типови од нивните неформални описи е опишан во [148]. Овој систем за изведување на формалната спецификација користи шеми, аналогија и расудување врз основа на разлики.

Исто така, постојат трудови во кои се користи интервална темпорална логика (ИТЛ) за опишување на софтверски и хардверски системи. Во [180], авторите прават формализирање на медицински водич преку проверка на квалитетот со користење на ИТЛ над природни броеви и го нарекуваат исказна логика на соседство. Овие водичи се документи кои ги користат лекарите при лекување на болест кај пациентите. Формален метод базиран на објекти за развој на системи во реално време е презентиран во [217]. Формалната спецификација на овој метод користи ИТЛ за опис на однесувањето на системот во реално време. Во [64], авторите користат ИТЛ и алатката Ana Tempura за да изградат и валидираат спецификациски модел за апстрактна трансакциска меморија (техника која нуди модел на паралелно програмирање за идните чип мултипроцесорски системи). Темпоралната логика се користи и за специфицирање на адаптивна семантика на програмите [219].

3.1.2 Придобивки од формалната спецификација

Формалната спецификација на апликациите овозможува математичка репрезентација на проблемот и подобрување на процесот на верификација. Едни од најважните придобивки од користењето на формални методи се следниве [78, 76, 190, 215, 163, 100, 46]:

- Формалниот опис на софтверот е апстрактен и прецизен опис што значи дека читателот може да ја разбере големата слика и сите недоследности може да бидат отстранети.
- Користењето на формалните методи допринесува за софтвер со поголем квалитет и софтвер со помалку грешки кои што може да бидат полесно откриени во споредба со стандардните методи за тестирање.
- Формалниот опис им дозволува на корисниците да прават строги анализи и да ги откријат корисните својства на системот, како и проблемите во спецификацијата на барањата.
- Формалните методи се применуваат за верификација која претставува обид да се докаже теорема дека ако одредени услови се задоволени, програмата ќе ги даде очекуваните резултати.
- Формалните методи го намалуваат времето на фазата на тестирање и одржување, ја зголемуваат довербата на развивачите на софтверот за коректност на софтверот, како и разбирливоста на системот.

Формалните методи задолжително се применуваат при развој на критични системи, но исто така постојат критични научни апликации, особено во сферата на медицина и биоинформатика. Ваквите апликации имаат критични импликации врз науките на живите организми, па значајно е да имаат потврда за квалитет [203].

3.1.3 Интервална темпорална логика

Интервална темпорална логика (ИТЛ) е формализам кој претставува продолжување на стандардната предикатна логика која вклучува временски зависни оператори [150]. Клучниот поим во ИТЛ е интервал. Интервал се дефинира како (бес)конечна низа на состојби, каде што секоја состојба преставува мапирање од множество на променливи во множество на вредности. [40].

Израз во ИТЛ се дефинира на следниот начин:

$exp ::= \quad z \mid a \mid A \mid g(exp_1, ..., exp_n) \mid \iota a : f,$

каде z е целобројна вредност, a е статичка променлива (нејзината вредност не може да се промени во даден интервал), A е променлива за состојба (нејзината вредност може да се промени во даден интервал), g е функциски симбол, f е формула.

Формула во ИТЛ се дефинира на следниот начин:

$f ::= \quad p(exp_1, ..., exp_n) \mid \neg f \mid f_1 \wedge f_2 \mid \forall v \cdot f \mid \text{skip} \mid f_1; f_2 \mid f^*,$

каде p е предикатен симбол, ; е операторот chop.

Формулите се градат индуктивно на следниот начин:

- Еднаквост: $exp_1 = exp_2$
- Логички сврзници: $\neg f$ and $f_1 \wedge f_2$
- Следно: $\bigcirc f$
- Секогаш: $\Box f$

Неформалната семантика на ИТЛ може да се претстави на следниот начин:

- $\imath a : f$ - избери вредност од a така што f важи
- skip - интервал со должина 1
- $f_1; f_2$ - интервалот се дели на два интервали (префикс интервал така што важи f_1 и суфикс интервал така што важи f_2) или интервалот е бесконечен и важи f_1.
- f^* - интервалот е делив во конечен број на интервали така што за секој од нив важи f, или интервалот е бесконечен и може да биде декомпониран во бесконечен број на интервали така што важи f [40].

Во продолжение е прикажана валидна ИТЛ формула: $(I = 2) \wedge \bigcirc(K = 3)$. Може да се интерпретира на следниот начин: моменталната состојба на I е 2, а во следната состоја K ќе има вредност 3.

Интервалната темпорална логика обезбедува основа за програмскиот јазик Темпура. Главните синтаксички категории во Темпура се: изрази (може да бидат рационални или аритметички), извештаи (временски формули кои можат да бидат едноставни или сложени) и локации (места каде што се чуваат вредности) [149]. Дадена формула ќе се изврши во Темпура доколку се исполнети следниве три карактеристики: формулата е детерминистичка, должината на временскиот интервал е позната и вредностите на променливите се познати на тој интервал [56].

3.1.4 Формална спецификација на код за пресметување на гранични состојби на потенцијал на Морзеов осцилатор

Во оваа секција е претставена формална спецификација на кодот за решавање на проблемот на наоѓање на гранични состојби на потенцијалот на Морзеов осцилатор, односно на решавање стационарна Шредингерова равенка за Морзеов потенцијал. Формалната спецификација на кодот се врши со користење на ITL и Ana Tempura. Резултатите од ова истражување се објавени во [128].

Гранични состојби на потенцијалот на Морзеов осцилатор

Пресметката на сопствените енергии на граничните состојби на Морзеовиот осцилатор е прототипска вежба во квантната механика. Ова е така бидејќи

потенцијалот служи како моделирачки систем за изучување на молекуларни вибрации. Морзеовиот потенцијал ја има следната форма:

$$U(r) = D_e \cdot [1 - exp(-\beta \cdot (r - r_e))]^2 \tag{3.1}$$

каде D_e ја означува дисоцијативната енергија на соодветната врска, додека r_e ie растојанието меѓу атомите кое одговара на минималната енергија на оцсилаторот. Вибрационата Шредингерова равенка со потенцијалот во форма (3.1) има аналитичко решение и соодветните вибрациони сопствени енергии се претставуваат со равенката :

$$E_v = h \cdot c \cdot [(v + \frac{1}{2}) \cdot \omega_e - (v + \frac{1}{2})^2 * \omega_e x_e] \tag{3.2}$$

Во равенката (3.2), v е вибрациониот квантен број ($v \in 0, 1, 2, \ldots$), $\omega_e x_e$ е таканаречената нехармонична константа и е поврзана со параметарот β и намалената маса на осцилаторот μ преку следната формула:

$$\beta = 2 \cdot \pi \cdot c \cdot \omega_e \cdot \sqrt{\frac{\mu}{2 \cdot D_e}} \tag{3.3}$$

Сите други симболи во (3.2) го имаат нивното вообичаено значење. Фактот дека вибрационата Шредингерова равенка за Морзеов оцилатор има аналитичко решение го прави системот прилично погоден тест случај за голем број на нумерички методи кои решаваат квантен вибрационен проблем.

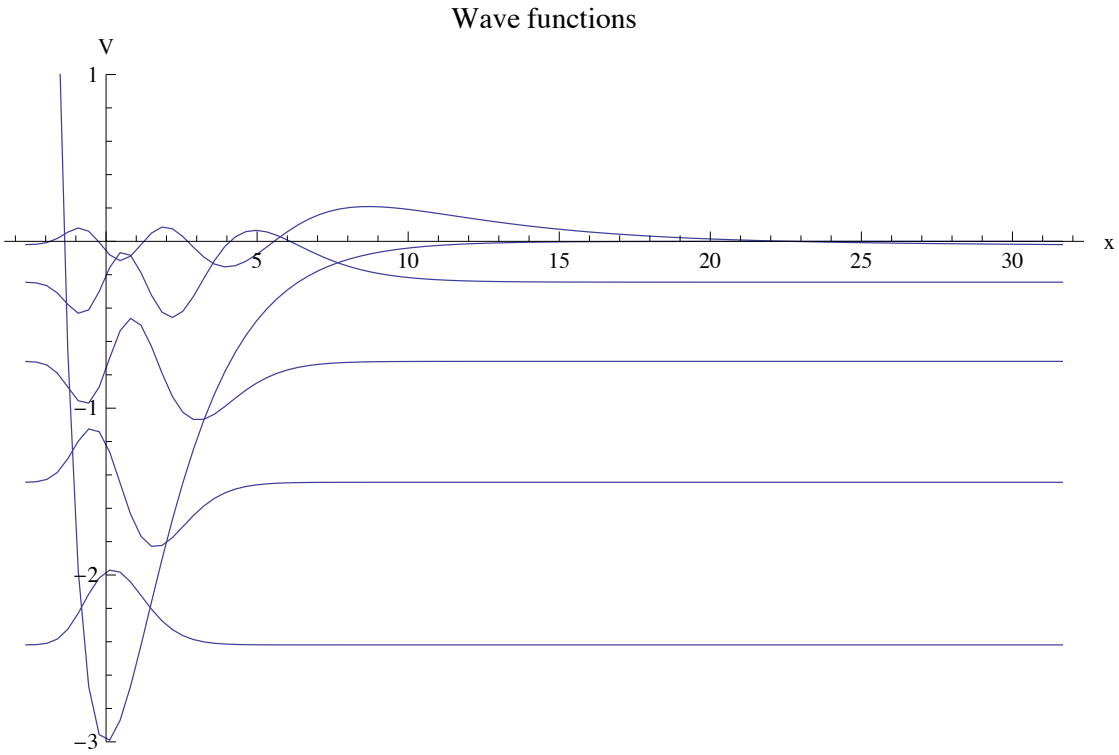
Во овој случај, примената на формална спецификација на код се врши на кодот за наоѓање на вибрациони сопствени вредности со користење на методологија на претставување на променливи со дискретни вредности [33], следејќи го пристапот на Битнер [32]. За да се реши проблем во квантната механика со користење на нумерички методи, Хамилтоновиот оператор треба да биде претставен користејќи конечна полиномна база. Во овој случај како база се користат полиномите на Чебишев.

Еден од методите за кои се прави формална спецификација е thcheby(...). Методот враќа множество од точки pts[NPTS], ke_fb[NPTS] (кинетичка енергија), множество од тежини w[NPTS] и матрица на трансформација T[NPTS] [NPTS]. NPTS е бројот на точки. Постојат две репрезентации: репрезентација со конечни бази (FBR) и репрезентација со дискретни прменливи (DVR). Матрицата на трансформација врши транформација од FBR во DVR.

```
double thcheby(double xmin, double xmax, double pts[],
double ke_fb[], double w[],double T[][NPTS])
{
double  del, fb;
int i,j;
del=xmax-xmin;
for(i=0;i<NPTS;i++)
{
   pts[i]=((i+1)*del)/(NPTS+1)+xmin;
   ke_fb[i]=square((i+1)*M_PI/del);
   w[i]=del/(NPTS+1);
```

```
    for(j=0;j<NPTS;j++)
        T[i][j]= sqrt(2.0/(NPTS+1))*sin(((i+1)*(j+1)*M_PI)/
        (NPTS+1));
}}
```

За решавање на граничните состојби на потенцилајот на Морзеовиот осцилатор, треба да се дефинира бројот на точки: NPTS= 100, и опсегот: $xmin = -3$ и $xmax = 32$. Целта е да се конструира Хамилтонова матрица во DVR основи и потоа да се изврши дијагонализација за да се одредат сопствените вредности и сопствените вектори. Сопствените вредности (енергиите) под 0 се гранични состојби. Сопствените вредности и сопствените вектори се користат за да се престават брановите функции (слика 3.1).



Слика 3.1 – Бранови фунцкии

Делот за проверување на коректноста на брановите функции преку проверка на вредностите на првите два сопствени вектори е целосно автоматизиран. Брановата функција е точна ако разликата меѓу две соседни вредности е помала од 10^{-3} и вредностите постепено се стремат кон 0.

3.1.5 ИТЛ формална спецификација со користење на програмскиот јазик Темпура

Формалната спецификација на код за пресметување на гранични состојби на потенцијалот на Морзеов осцилатор е реализирана со пишување на код во

Темпура и креирање на тестови за да се споредат резултатите на излез од извршната верзија на програмата напишана во програмскиот јазик C и кодот во Темпура.

Со цел да се воспостави комуникација помеѓу датотеката во Темпура и извршната верзија на програмата, мора да се вметнуваат предикати во изворниот код во програмскиот јазик C. Во продолжение е даден кодот за проверка на вредностите на низата ke_fb[]. Кодот за проверка на вредностите на останатите низи е сличен.

```
int i=0;
double ke_fb_i=0.00000;
assertion1 ("ke_fb_i", ke_fb_i);
assertion ("i", i);
while (i<NPTS) {
 ke_fb_i=square((i+1)*M_PI/del); assertion1 ("ke_fb_i", ke_fb_i);
 i=i+1;assertion ("i", i);
     }
```

Предикатните функции се користат за проверка на вредностите на променливите после секоја направена промена. Вредностите на променливите се споредуваат со вредностите добиени од кодот напишан во Темпура.

Во продолжение е дадена функцијата за пресметување на Y-тата вредност на низата ke_fb[] напишана во Tempura. Функцијата itof враќа децимален број кој кореспондира на соодветниот цел број и притоа $del = xmax - xmin$.

```
define calc_ke_fb(Y) = {
 if Y>=0 then {((itof(Y)+$1.0$)*M_PI/del)*((itof(Y)+$1.0$)*M_PI/del)
}
 else empty
}.
```

За споредба на вредностите добиени од кодот напишан во програмскиот јазик C и кодот во Темпура, се дефинира следнава функција:

```
define check(Z,X,Y) = {
   {
      skip and get_var_float("ke_fb_i",X) and stable(X) and

        if fabs(X-Z)<$0.000001$ then {

       format("!:: Pass ke_fb_i Prog %f Prop %f\n", X, Z)
      } else {

       format("!:: Error: Prog %f Prop %f\n", X, Z)
      }
   };
```

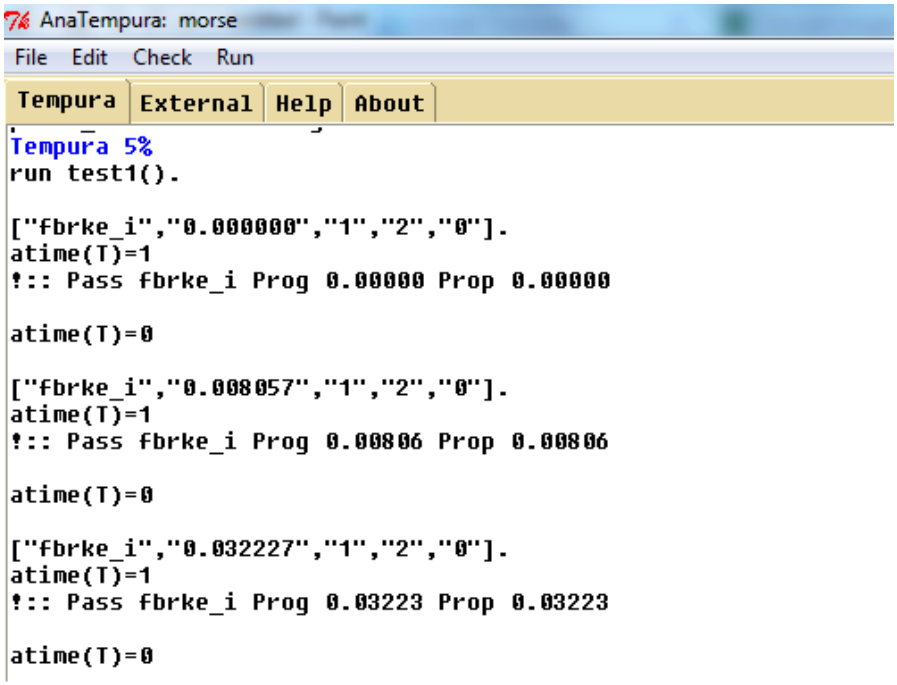
```
    { skip and get_var("i",Y) and  stable(X) and stable(Y) };
    {
       skip and  stable(Y)
          }
}.
```

Функциите get_var и stable се користат за проверка на вредностите добиени од програмата напишана во C (на оние места кои се додаваат предикатите). На сличен начин ја дефиниравме и функцијата floats get_var_float.

Во продолжение е даден тестот за проверка на резултатите:

```
/* run */ define test1() = {
 exists  ke_fb_i, Y :
{
check($0.00000$,ke_fb_i,Y);
while (Y<NPTS) do
{
check(calc_ke_fb(Y),ke_fb_i,Y)
}
}
}.
```

Функцијата check ги споредува резултатите од функцијата calc_ke_fb во Темпура и програмата во C. Доколку резултатите се идентични, велиме дека тестот поминал. Резултатите од првите две итерации кога тестот се извршува во алатката Ana Tempura се прикажана на слика 3.2.



Слика 3.2 – Извршување на test1 во Ana Tempura

3.2 Градење на научни работни текови

3.2.1 Карактеристики на научните текови

Во согласност со стандардот ISO 12.651-2, поимите работен тек, придвижувачка сила на работен тек и систем за управување со работни текови се дефинирани како [11, 10]:

- Работен тек - автоматизација на процес, во целост или делумно, за време на кој се пренесуваат документи, информации или задачи од еден од друг учесник во акцијата, според множество на процедурални правила.
- Придвижувачка сила на работниот тек - софтверски сервис кој обезбедува околина за извршување на работниот тек.
- Систем за управување со работни текови - систем со кој се дефинира, создава и управува со извршувањето на работните текови преку употреба на софтвер кој е во состојба да ја толкува дефиницијата на процесот, да воспостави интеракција со корисниците, и каде што е потребно да повика ИТ алатки и апликации.

Научните работни текови се разликуваат од бизнис работните текови. За разлика од бизнис работните текови кои се првенствено ориентирани кон моделирање на контрола на проток на текот, научните работни текови се наменети за обликување на научни процеси со големо множество на податочно интензивни компјутерски процеси [66]. Главната цел на научните работни текови е да се автоматизира научниот процес каде што задачите ќе бидат структурирани врз основа на нивната контрола и зависност од податоци [216].

Целта на научните работни текови е да се обезбеди едноставна и концизна нотација која овозможува паралелизација на задачи и пресметки кои се потребни во дадената научната област. Научните работни текови мора да овозможат независните задачи да се извршуваат паралелно и да бидат распределени за извршување во кластерирани мрежни околини [220].

3.2.2 Досегашни истражувања

Barker и van Hemert [24] прават истражување за постоечката технологија за работни текови од бизнис и научни домени и предлагаат некои решенија за иден развој на системи за научни работни текови. Bharathi и останатите [29] ги опишуваат основните работни структури кои се вклучуваат во сложените работни текови креирани од научните заедници. Тие исто така прикажуваат карактеристики на работни текови од пет научни апликации, вклучувајќи го и нивниот состав, податоци, и пресметковни барања. Ramakrishnan и Gannon [168] презентираат анкета за дистрибуираните карактеристики на работните текови и потребите за ресурси. Тие разговараат за работни примери од различни области: биоинформатика и биомедицина, моделирање на временската прогноза и океани, астрономија, итн. Deelman и останатите [58] извлекуваат таксономија на карактеристики од начинот на кој научниците ги употребуваат

постоечките системи на работни текови и даваат примери земени од постоечки систем за работни текови.

Wolstencroft и останатите [213] ја опишуваат алатката за градење на работни текови Taverna. Според нив, Taverna е алатка дизајнирана за комбинирање на дистрибуирани веб сервиси и/или локални алатки. Авторите, исто така, велат дека една од главните карактеристики на овие системи е дека работните текови може да се искористат повторно. Goble и De Roure [74] ја презентираат работата на Taverna и потенцираат дека околините за градење на научни работни текови побаруваат богати алатки кои ги поддржуваат експериментите направени од страна на научниците. Пример за тоа е myExperiment кој претставува иницијатива да се создаде социјална мрежа од дизајнери на работни текови. Oinn и останатите [155] даваат опис на Taverna и ги презентираат научените лекции во текот на својата кариера. Нивната цел е да се покаже како работните текови се вклопуваат во експерименталниот контекст на научниците.

Reuillon и останатите [172] го претставуваат доменски специфичниот јазик OpenMOLE преку пример на модел на играчка и преку автоматска калибрација на комплексен модел на систем од реалниот свет во областа на географијата. Истиот автор го опишува OpenMOLE како научна рамка која обезбедува виртуелизирана извршувачка околина за дистрибуирано пресметување [171].

Споредба помеѓу двата системи за градење на работни текови, Taverna и BioFlow е претставена во [103]. Авторите сакаат да покажат дека декларативниот дизајн на работни текови со користење на BioFlow е поефикасен и поевтин, отколку традиционалните пристапи со користење на системи како Taverna. Во својот труд, Tan и останатите автори, [199] вршат споредба на користење на алатката за градење на научни работни текови Taverna и јазикот на извршување на работни текови BPEL. Ја споредуваат употребливоста, вклучувајќи го составот на работните текови, извршувањето и анализата на резултатите. Нивното истражување покажува дека BPEL како императивен јазик нуди сеопфатно множество на моделирачки компоненти за работните текови, додека Taverna нуди модел на проток на податоци и покомпактно множество на компоненти кои помагаат за моделирање на проток на податоци и извршување. Curcin и Ghanem [54] предлагаат рамка за споредување на научни и бизнис системи за градење на работни текови. Рамката е базирана на контрола и својства на протокот на податоци. Тие прават преглед на шест системи за градење на научни работни текови : Discovery Net, Taverna, Triana, Kepler, Yawl и BPEL.

3.2.3 Алатки за градење на научни работни текови

Алатката Taverna

Taverna [213] е доменско независен систем за управување со работни текови наменет за дизајнирање на научни експерименти. Достапна е како десктоп клиентска апликација (Taverna Workbench), Taverna сервер кој овозможува далечинско извршување на работниот тек и извршување од командна линија. Taverna е создадена од тимот на myGrid и е интегрирана со две други алатки на myGrid : myExperiment (околина за споделување на работни текови) и

BioCatalogue (каталог на веб сервиси за природните науки). Работниот тек креиран во Taverna може да биде дистрибуиран на грид, облак, или во комбинација со други производи. Последната верзија е објавена во 2014 година.

Алатката Taverna е скалабилен систем за градење на научни работни текови кој има пристап до локални и оддалечени ресурси и грид сервиси (повеќе од 3500). Taverna има поддршка за повикување на клиентски библиотеки (напишани во Ruby и Java), како и алатки и скрипти поставени на локални или машини на растојание. Исто така, има поддршка за табели во CSV и Excel формат. Една од најинтересните карактеристики на Taverna е тоа што овозможува брза инкорпорација на нови сервиси без кодирање. Има графички дизајнер за работни текови кој работи на принцип „повлечи и пушти". Taverna обезбедува валидација на работните текови и преглед на вредностите за време на извршувањето. Најголема примена има во следните области : биоинформатика, хемија, астрономија, инженерство, мултимедија, општествени науки и многу други.

Овој систем за управување со работни текови им овозможува на корисниците да дефинираат каков ќе биде протокот на податоците помеѓу сервисите, но тие не треба да се грижат како ќе се повикаат сервисите. Податоците, исто така, можат да се конвертираат од еден во друг формат. Резултатите од експериментите може да се следат со помош OPM (Open Provenance Model) стандардот. Кога работниот тек ќе заврши може да се сподели на myExperiment и да биде пребаран и преземен од други корисници.

Taverna има вградена поддршка за додавање:

- Сервиси - Taverna поддржува користење на различни видови на сервиси: WSDL веб сервиси (треба да се внесе само URL адресата), BioMoby веб сервиси, BioMart веб сервиси, SoapLab веб сервиси, локални Јава веб сервиси (Beanshell скрипти), локални Java API, R скрипти на R сервер (RShell скрипти - анализи добиени употреба на статистичкиот пакет R), XPath, OAuth сервиси, сервиси за компоненти, текстуални константи (за поставување на фиксна вредност за влезни параметри на сервисиот).

  Сите сервиси кои може да бидат пристапени од страна на Taverna може да се поделат во две категории: доменски сервиси (често овозможени од страна на добавувачи (трети лица) и извршуваат научни функции и сервиси-подлоги (за поврзување на влезови и излези со слични сервиси). Корисниците исто така можат да дефинираат сопствени сервиси.

- Листи и итерации - Сервисите во Taverna како резултат можат да вратат една вредност или листа на вредности или листа од листи. Секоја листа има длабочина (грануларност на порта: 0 - една вредност, 1 - листа од вредности, 2 - листа од листи, итн.) Доколку сервисите имаат различни длабочини, тогаш ќе се направи имплицитна итерација (ќе се создаде нова листа од резултатите на сервисот).

- Јамки - вообичаено се користат за повикување на асинхрони услуги - веб сервис или слично, каде што треба да се повтори било каква акција.

- Контролни врски - се користат за поставување на зависностите помеѓу сервисите кои не споделуваат директно податоци. Се одложува повикување на сервисот додека не заврши извршуањето на претходниот.

- Спојувања - овозможуваат комбинација на резултати од повеќе сервиси во еден единствен влез. Ќе се создаде листа на сите излези од сервисите.

- Паралелни повици на сервиси - се користат кога ист сервис се повикува повеќе пати во исто време. Сервисот ќе се повика штом потребните влезови на итерација се на располагање, но не повеќе од специфицираниот број на итерации.

- Компоненти - реискористлива функционална единица дефинирана во работните текови. Портите и сервисите имаат семантички анотации.

Taverna поддржува неколку безбедносни стандарди: поддршка за HTTP основна автентикација, веб сервиси за безбедност, HTTPS и Java политика на безбедност. Исто така, за Taverna има развиено голем број на додатни компоненти.

Алатката OpenMole

OpenMole (Open MOdeL Experiment) [173] системот за градење на работни текови нуди околини за паралелно извршување на природно паралелни процеси. Развиен е во рамките на слободната софтверска лиценца AGPv3 [71]. Последната верзија на софтверот 5.9 е од 15 ноември, 2015 година. Овој систем за градење на научни работни текови поддржува напредни нумерички експерименти при симулација на модели и дистрибуирање на работните текови на машини со повеќе јадра, кластери и грид. Корисниците имаат можност да вклучат Јава датотеки, бинарни извршни датотеки или NetLogo извршни датотеки. Корисниците не треба да се грижат за големината на податоците и задачите бидејќи OpenMole е скалабилен и поддржува терабатји од податоци и милиони задачи.

OpenMole нуди графички интерфејс и командна линија за дизајнирање експерименти и нивна дистрибуција. Работниот тек во OpenMole се вика Mole. Тој може да содржи задачи, транзиции, прототипови, примероци, околини, приклучоци и извори. За да се изврши, работниот тек мора да содржи најмалку една задача и почетна задача (капсула). Опис на компонентите на еден работен тек се дадени подолу:

- Прототип - променлива активна за време на работниот тек. Содржи три карактеристики: име, тип (цел број, текст, итн.), димензија (0 - скалар, 1 - вектор, 2 - матрица, итн.). Прототипот мора да биде дел од една задача (влез или излез).

- Задача - постојат неколку видови на задачи: Едноставна (Groovy), Истражувачка (Exploration), SystemExec, задача на работниот тек (Mole), Моделирачка задача базирана на агенти (Agent based model task), задача за сензитивност (Sensitivity), статистичка задача (Stat). Секој од нив содржи име, простор за влезни и излезни податоци (за примање или

испраќање на прототип на друга задача), околина (за доделување на компјутерски ресурси на дадена задача - грид, кластер, ..). Исто така, постои и валидациска задача која проверува дали влезот/излезот е валиден, дали сите задолжителни полиња се пополнети, итн. Деталите за секој тип на задача се дадени во продолжение:

  - Едноставна (groovy) задача - извршува некои мали делови на скрипти со код или повикува код кој се извршува на JVM (Java, Scala ..) со поставување на .jar архива и со повик на методи .
  - Истражувачка (exploration) задача - овозможува вклучување на примерок со што ќе се создаде низа на комбинации од прототипи за време на извршување. Тоа значи дека сите задачи кои треба да се извршат по истражувачката задача ќе се реплицираат со различни вредности на прототипот.
  - SystemExec задача - за извршување на C, C ++ и Python кодови.
  - Задача на работниот тек (Mole) - извршува друг работен тек. Ова овозможува вградување на работен тек во задачата. Корисно е и кога комплексен работен тек треба да се подели на помали работни текови.
  - Моделирачка задача базирана на агенти (Agent based model task), - овозможува да се вгради NetLogo модел [208] и автоматски ги мапира влезовите и излезите кои ќе ги најде во .nlogo Globals.
  - Статистичка задача (Stat) - пресметува просек, сума и средна вредност на прототип со димензија 1.
  - Задача за сензитивност (Sensitivity) - овозможува метод базиран на варијанса да го утврди влијанието на параметрите и/или варијаблите на варијансата на излезот(ите) на моделот.

- Транзиција - стрелка која го дефинира редот на приоритети помеѓу задачите. Може да се конвертира во: агрегирачка транзиција - доаѓа по истражувачката задача и обезбедува спектар на прототипови; транзиција преку канал на податоци - само овозможува прототипот да помине, не ја предизвикува следната задача.

- Креирање на примероци - примерокот може да биде создаден графички. Постојат два вида на ентитети: домен - како променливата може да варира; креирање примероци - како домени и други примероци можат да се состават и разделат. Постојат многу видови на примероци: комплетен, мешан, комбиниран, итн. и домени: опсег, повеќе датотеки, една датотека, униформна дистрибуција, итн.

- Приклучок - „слушател" на работниот тек. Врши одредена акција на излез. Постојат неколку видови на приклучоци: приклучок за прикажување на прототип (приказ на стандарден излез на секоја вредност на прототипот кој го слуша); приклучок за копирање на датотеки (зачувува

прототип во локален директориум); приклучок за додавање на прототипови на датотеки (секоја вредност на прототипот кој се слуша се додава во една датотека).

- Извор - чита податоци од CSV датотека и ги мапира во прототиповите вклучени во работниот тек.

Споредба на OpenMole и Taverna

Двете алатки имаат многу сличности во однос на функционалноста, но се разликуваат малку во начинот на имплементација на задачите. Taverna нуди голем број на функции, како што се додавање на постоечките веб сервиси, автоматска конверзија на длабочина на листата со цел да се овозможи поврзување на два сервиси. Од друга страна, OpenMole нуди поедноставна дефиниција на композитни влезови (на пр. правење на комбинации од елементи во низа). Во табела 3.1 е прикажана споредба на некои од важните карактеристики на овие алатки.

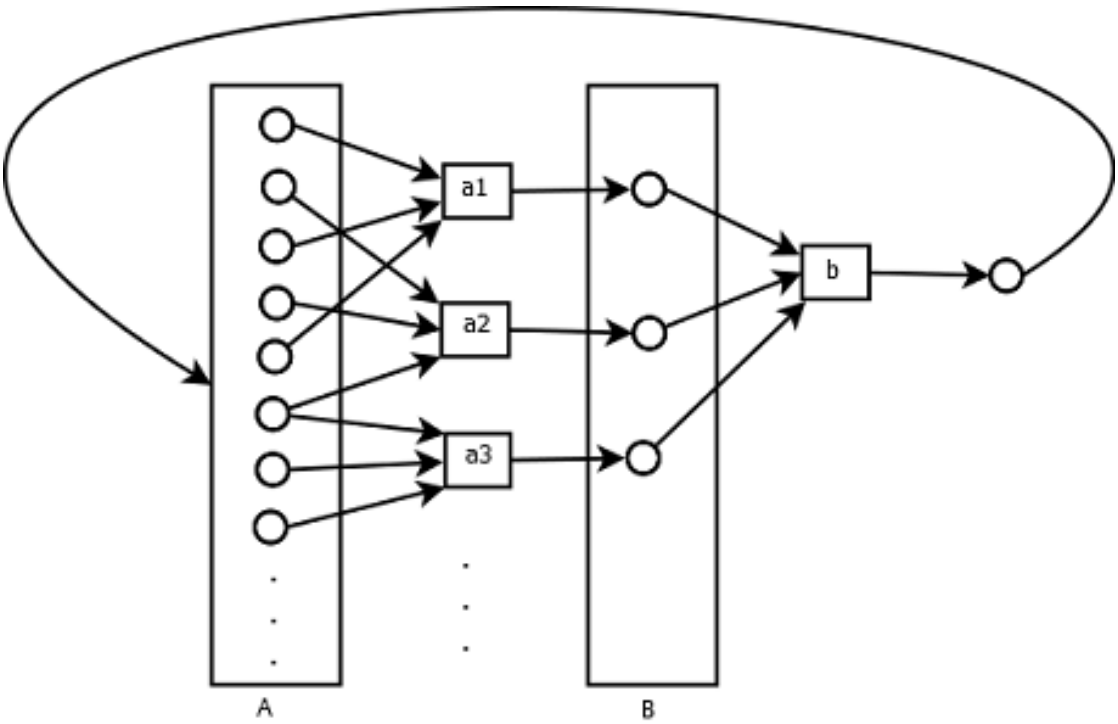
Табела 3.1 – Споредба на карактеристики на OpenMole и Taverna

Карактеристика	OpenMole	Taverna
Графички интерфејс за дизајн на работни текови	Y	Y
Метод на повлечи и пушти	N	Y
Скалабилност	Y	Y
Дистрибуција на далечни машини во грид	Y	Y
Композитен влез - примероци (графички)	Y	N
Повикување на онлајн веб сервиси преку URL	N	Y
Околина за паралено извршување	Y	Y
Автоматско паралелно извршување на задачи	N	Y
Поддршка за безбедносни стандарди	N	Y
Автоматска конверзија на длабочините на влез/излез	N	Y
Контролни линкови	N	Y
Различна длабочина на влез и излез	Y	Y
Вгнездени работни текови	Y	Y
Валидација на работни текови	Y	Y
Вградување на Java извршни датотеки	Y	Y
Извршување на код од скрипта	Y	Y
Вградување на NetLogo модел	Y	N
Итерирање	Y	Y
Спојување на излези	Y	Y
Читање на податоци од CSV датотеки	Y	Y
Креирање на компоненти	N	Y
Можност за приклучоци	Y	Y
Следење на резултати	Y	Y
Поддршка за споделување на работни текови	N	Y
Поддршка за пребарување на работни текови	N	Y

3.2.4 Имплементација на работен тек

Во оваа секција е претставена имплементација на работен тек во двете алатки OpenMole и Taverna. Резултатите од ова истражување се објавени во [118].

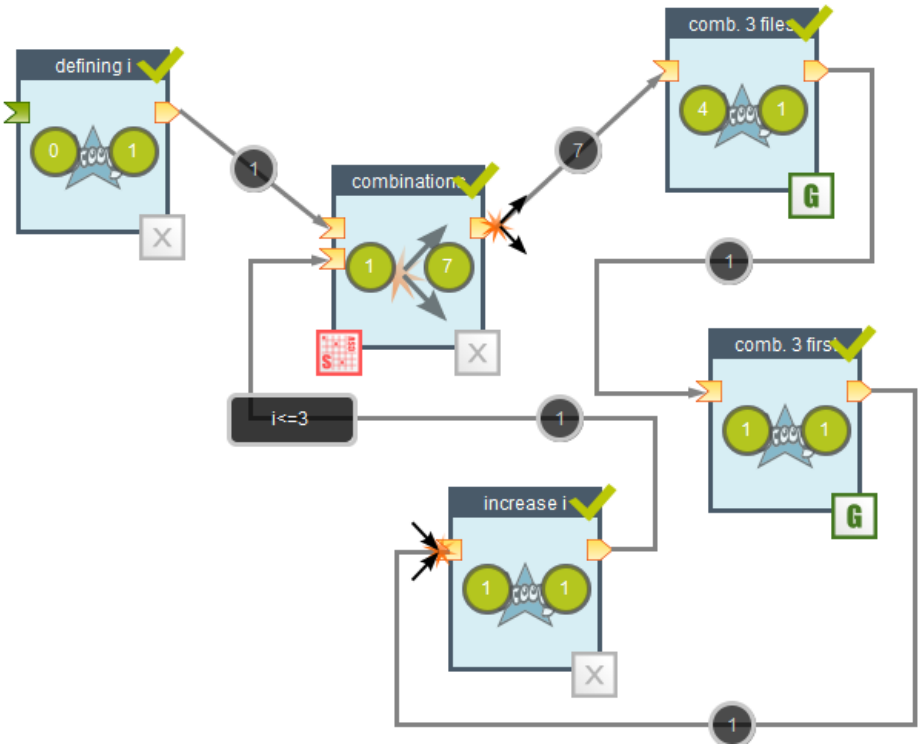
Работниот тек е прикажан на слика 3.3. Секој круг претставува датотека која содржи одредени податоци. Правоаголниците $a_1, a_2, a_3.., a_{64}$ означуваат акции врз датотеките, во нашиот случај спојување на содржината на датотеките. Секогаш се избираат 64 комбинации од 3 датотеки. За да се реализира тоа, се прават три копии од листата на датотеки во директориумот „A". Елементите од секоја листа од датотеки се мешаат произволно и се избираат четири датотеки од сите три листи. Правоаголникот „A" е директориумот каде се избираат датотеките и претставува почетна точка на извршување во работниот тек. Резултатот од секоја акција $a_1, a_2, a_3, ..a_{64}$ е датотека зачувана во директориумот „B". Акциите $a_1, a_2, a_3, ..a_{64}$ се извршуваат паралелно. Исто така, кога ќе се креираат три датотеки во директориумот „B", се извршува акцијата b која ја спојува содржината на трите датотеки во една датотека. Потоа, се бришат датотеките од директоримот „B". Акцијата b се повторува $64/3 = 21$ пат и се извршува паралелно со акциите $a_1, a_2, a_3, ..a_{64}$, веднаш после креирањето на три нови датотеки во директориумот „B". Кога ќе се креира нова датотека како резултат на акцијата b, истата се преместува во директориумот „A". Работниот тек може да се извршува во итерации, во зависност од корисничкиот влез.



Слика 3.3 – Пример на работен тек

Имплементација на работниот тек во OpenMole

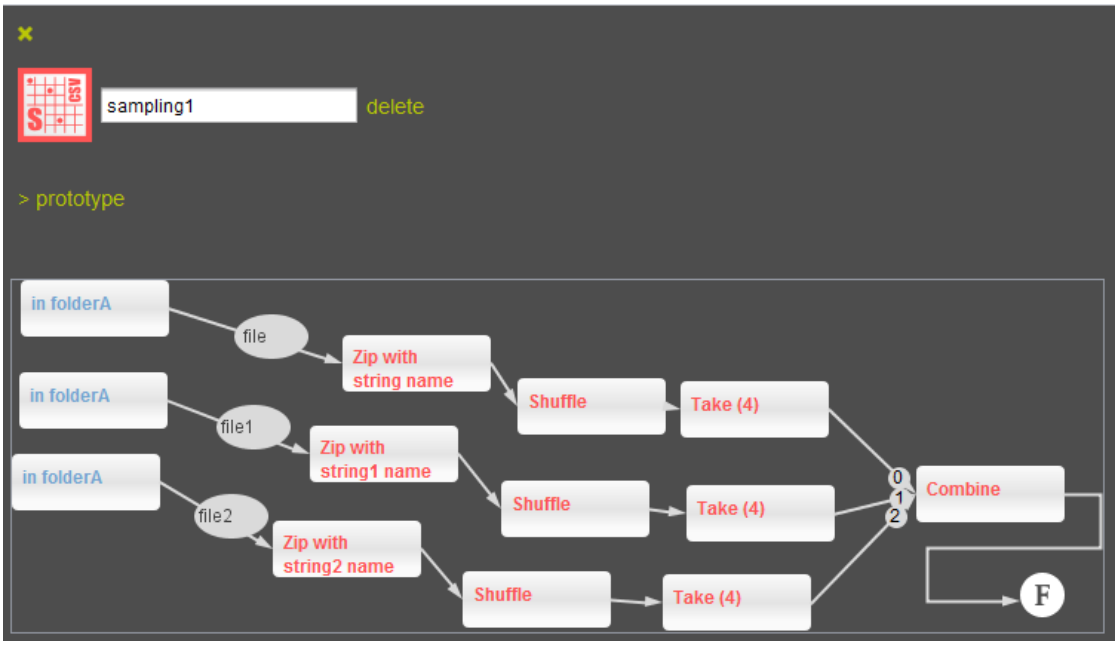
Имплементацијата на работниот тек во OpenMole е прикажана на слика 3.4. Почетната „defining i" капсула е едноставна (groovy) задача и се користи за поставување на прототипот од цел број i на 1. Прототипот i се користи како бројач за бројот на итерации на работниот тек. Излезот на оваа капсула е само i.



Слика 3.4 – Имплементација на работниот тек во OpenMole

Капсулата „combinations" е истражувачка задача и се користи за дефинирање на примерокот претставен на слика 3.5. Најпрвин, се дефинираат три датотечни домени „folderA" и три прототипови „file", „file1" и „file 2". Трите домени имаат иста патека на директоримуот (директориумот „A" прикажан на сликата 3.3). Примероците: „Zip with string name", „Zip with string1 name" и „Zip with string2 name" се користат за создавање на низи од имиња на датотеки во директориумот „A". По креирањето на трите низи, елементите од секоја од нив се мешаат произволно користејќи го примерокот за мешање (Shuffle sampling) и се избираат само првите 4 елементи (Take (4) Sampling). Четирите имиња на датотеки од секоја низа се комбинираат (Combine sampling) во вкупно 64 комбинации ($4 * 4 * 4$). Има 7 излези од оваа задача : прототипот i и 6 низи (file[1], file1[1], file2[1], string[1], string1[1] и string2[1]).

Следната капсула „comb. 3 files" е едноставна (groovy) задача и вклучува jar датотека. Претходната и оваа задача се поврзани со истражувачка (exploration) транзиција, што значи дека истиот метод ќе се изврши паралелно на различни комбинации од три датотеки во директориумот „folderA" ($a1, a2, a3, ...$ $a64$) како што е прикажано на Слика 3.3. Повикуваме метод од .jar датотеката



Слика 3.5 – Дизајнирање на примероци во OpenMole

што прима три текстуални низи како влезни параметри. Во нашиот случај, тоа се имиња на датотеки од секоја комбинација. Содржините на трите датотеки ќе се спојат во една датотека и таа ќе биде зачувана во директориумот „folderB". Излезот од оваа капсула е само целобројниот прототип i.

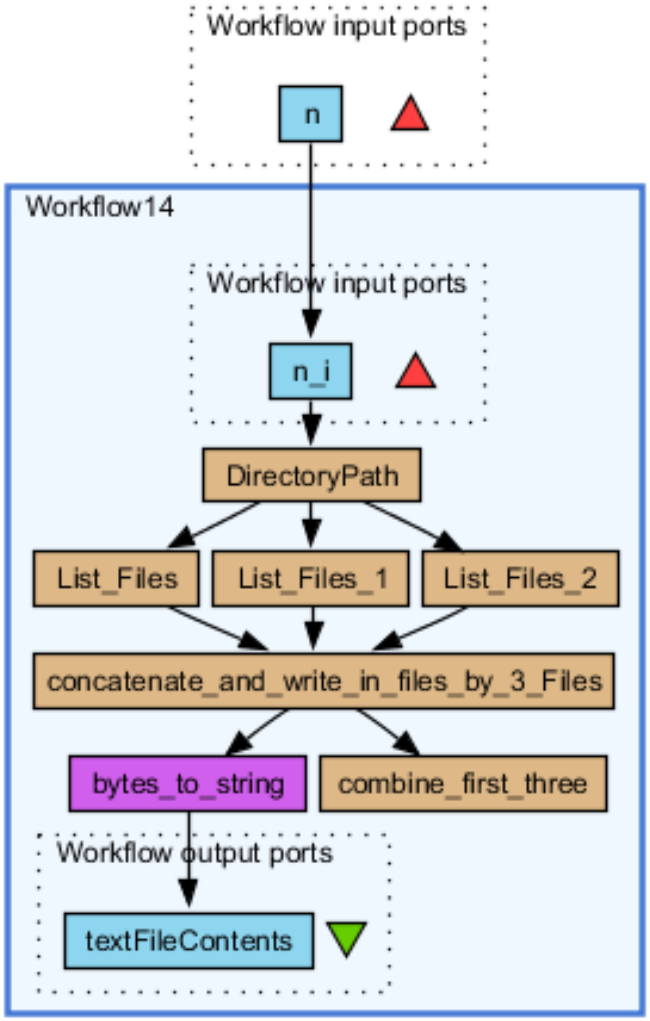
Капсулата „comb. 3 first" е исто така едноставна (groovy) задача и вклучува jar датотека. Методот повикан од .jar датотеката се користи за да провери дали трите датотеки постојат и да ги спои во директориумот „folderB". Оваа задача (b) се извршува паралелно со претходната задача „comb. 3 files". Кога ќе се креираат три датотеки, нивните содржини се спојуваат во нова датотека и таа се преместува во директориумот „folderA". Трите датотеки се бришат од директориумот „folderB". Излезот од оваа капсула и исто така само целобројниот прототип i. Тука може да бидат додадени различни типови на приклучоци. На пример, ако сакаме да ги видиме имињата на новите датотеки, треба да ги додадеме имињата како излез на капсулата.

Последната капсула во овој работен тек „increase i" е едноставна (groovy) задача и се користи само за зголемување на целобројниот прототип i. Транзицијата помеѓу капсулата „comb. 3 first" и оваа капсула е агрегација бидејќи оваа задача не треба да се извршува повеќе пати, туку само еднаш во секоја итерација од работниот тек. Задачата е поврзана со втората задача само за контролирање на вредноста на целобројниот прототип i. Условот за оваа транзиција е $i \leq 3$.

За извршување на задачите на Грид, потребно е да се креира околина за Грид и истата да се додаде како околина за извршување на капсулите „comb. 3 files" и „comb. 3 first".

Имплементација на работниот тек во Taverna

Имплементацијата на работниот тек во Taverna е прикажана на Слика 3.6. Работниот тек „Workflow 14" претставува вгнезден работен тек. Тоа овозможува работниот тек да биде извршен n пати. Влезниот параметар n е листа од целобројни вредности. Ова претставува почетна точка на извршување и единствената значајна работа е бројот на елементи во листата. Исто така, на почетокот на внезденият работен тек постои влезна порта n_i - целобројна вредност која се зголемува во секоја итерација на работниот тек (итерира низ елементите во листата).



Слика 3.6 – Имплементација на работниот тек во Таверна

Сервисот „DirectoryPath" преставува beanshell скрипта каде патеката на

директориумот „folderA” е специфицирана како текстуална низа од знаци "directory”. Ова е единствениот излез од скриптата. Следно, постојат три сервиси „List_Files”, „List_Files_1”, „List_Files_2” кои се идентични, но се користат за создавање на истата низа од датотеки од влезниот директориум. Потоа, елементите од секоја од трите низи од датотеки се мешаат произволно и се избираат само првите четири елементи од листите. Ова се прави со пишување на скрипта (Java код). Овие три сервиси ќе креираат 64 произволни комбинации, секоја од по три датотеки. Излезот на секој од овие три сервиси е листа од четири датотеки.

Следниот сервис, „concatenate_and_write_in_files_by_3_Files” има влезни параметри со должина 0, што значи дека зема еден по еден елемент од секоја од трите листи од четири датотеки. На овој начин се прават сите комбинации. Овој сервис исто така е beanshell скрипта и ги спојува содржините на секоја комбинација од три датотеки. Оваа акција се повторува 64 пати $(a1, a2, a3, ... a64)$. Секоја новосоздадена датотека се зачувува во директориумот „folderB”.

„Combine_first_three” е сервис кој проверува дали постојат трите датотеки и ги спојува во нова датотека. Овој сервис (b) се извршува паралелно со претходниот сервис „concatenate_and_write_in_files_by_3_Files”. Кога ќе се креираат три датотеки, нивните содржини се спојуваат во нова датотека и таа се преместува во директориумот „folderA”. Трите датотеки се бришат од директориумот „folderB”.

Сервисот „bytes_to_string” е локален сервис за обработка на текст кој што конвертира низа од бајти во низа од текстуални карактери. Овој сервис се користи само за прикажување на излезот од секоја новосоздадена датотетка (вкупно 64 од секоја итерација на работниот тек).

За да се изврши на Грид, работниот тек мора да биде инсталиран како апликација на некој од јазлите од гридот и потоа да биде извршен на далечина користејќи ја алатката за пишување на командна линија од Taverna или Taverna Server.

3.3 Примена на програмски модели за решавање на проблеми од научен домен

3.3.1 Моделот MapReduce

MapReduce е програмски модел за паралелна обработка на големи множества на податоци [144]. Поделбата на влезните податоци и распределбата на извршување на програмата на неколку машини се одговорностите на системот за времето на извршување. Корисникот треба да специфицира мапирачка функција (Mapper), која ги обработува паровите клуч-вредност и редуцирачка функција (reducer) која ги спојува сите меѓу-вредности кои имаат ист меѓу-клуч. [57].

Моделот на MapReduce може да се подели во етапи, каде што секоја етапа содржи три фази: Мапирање, Мешање и Сортирање и Редукција. Фазата

на мапирање го мапира секој пар (клуч, вредност) на машините во системот како множество на парови (клуч, вредност), каде што вредноста на секој нов пар е подниза на оригиналната вредност. Фазата на мешање е одговорна за сортирање и пренесување на резултатите од мапирањето до редукторите. Фазата на редукција извршува некоја функција над податоците на секоја машина [207].

Програмата во која е имплементиран MapReduce моделот се состои од конечна низа на етапи специфицирани како двојки (торки од два елемента), каде што секоја торка содржи мапирачка и редуцирачка функција. Формално, ова може да биде запишано на следниот начин: $((M_1, R_1), (M_2, R_2), ..., (M_n, R_n))$ каде M_i е мапер, R_i е редуктор, i е цел број и $1 \leq i \leq n$. Двојка се дефинира како (M_i, R_i). Нека влезот во програмата кој е множество од повеќе парови (клуч;вредност) биде означен со U_0 и излезот кој е множество од повеќе парови (клуч;вредност) на i-тата етапа нека биде означен со U_i.

Програмата се извршува за $r = 1, ...n$. За секое r, се извршуваат фазите на мапирање, мешање и редукција. Фазата на мапирање го предава парот (клуч;вредност) $(k; v)$ во U_{r-1} на маперот M_r и го извршува. Излезот од маперот M_r е низа од (клуч;вредност) парови $(k_1; v_1)$, $(k_2; v_2)$,... и може да биде дефинирана на следниот начин: $U'_r = \cup_{(k;v)\in U_{r-1}} M_r((k; v))$. Фазата на мешање конструира $V_{k,r}$ (вредности така што $(k; v_i) \in U'_r$) од U'_r за секое k. Фазата на редукција го предава k и некоја произволна пермутација на $V_{k;r}$ на посебна инстанца од редукторот R_r и ја извршува за секоја k. Излезот од редукторот е низа од двојки $(k; v'_1)$, $(k; v'_2)$,... и U_r така што множеството од парови (клуч; вредност) генерирани од редукторот R_r се дефинира како $U_r = \cup_k R_r((k; V_{k,r}))$ [191, 108].

Програмите кои го користат MapReduce моделот ги имплементираат интерфејсите Mapper и Reducer за да овозможат мапирачка и редуцирачка функција. Методите за мапирање и редукција може да се претстават како што е прикажано подолу. Вредностите со ист клуч се редуцираат заедно [158].
метод Мапирај(клуч k, вредност v) → ЕМИТИРА(клуч k', вредност v')
метод Редуцирај(клуч k, вредност v) → ЕМИТИРА(клуч k', вредност $[v', v_2, v_3$ ...]

Моделот MapReduce автоматски поддржува паралелно програмирање и го заштитува програмерот од пишување код за дистрибуција на податоци, распределување и толеранција на грешки. Програмерот само треба да ги определи функциите за мапирање и редукција. Ова, исто така, може да се смета како недостаток на моделот бидејќи програмерот не може да влијае на ефикасноста на паралелизмот. Тоа значи дека сите видови на проблеми не се погодни да се решат со помош на моделот MapReduce [108].

Обработката на научни податоци и кластерирачки алгоритми кои се користат во областите на хемија, биологија, физика, се компјутерски интензивни операции и користењето на техники за паралелизација е клучно за да се постигне ефикасна анализа на податоците. MapReduce моделот е погоден кога обработката на податоците треба да се подели на помали независни пресметки и меѓу-резултатите треба да се спојат со некоја пост-обработка со цел да се добие конечниот резултат. Со тоа се обезбедува едноставност, робусност и има помалку ограничувања околу синхронизацијата која носи дополнителни

трошоци [63].

Hadoop (Apache Hadoop) е софтверска библиотека со отворен код за обработка на податоци што овозможува дистрибуирано и паралелно процесирање на големи множества податоци. Проектот Hadoop вклучува четири различни модули: Hadoop основни услуги (услуги кои ги поддржуваат другите Hadoop модули), Hadoop дистрибуиран датотечниот систем (дистрибуиран датотечен систем кој обезбедува брз пристап до податоци), Hadoop YARN (рамка за распоредување на процеси и управување со ресурсите на кластерот) и Hadoop MapReduce (систем базиран на YARN за паралелно процесирање на големи множества на податоци) кој се користи од страна на многу компании како што се Facebook, Cloudera, Amazon, Microsoft, Yahoo, итн. Обработката на податоци во Hadoop може да се спроведе со MapReduce директно или преку користење на јазици на високо ниво и преведување во MapReduce процеси подоцна [206, 17]. Hadoop може да се користи за создавање на складишта на податоци. Пример за тоа е HIVE кој поддржува креирање и извршување на прашалници напишани во HiveQL (декларативен јазик што наликува на SQL). Прашалниците се компајлираат во MapReduce процеси и се извршуваат на Hadoop [200].

3.3.2 Досегашни истражувања

Постојат повеќе трудови во кои MapReduce моделот се користи за решавање на проблемите во научниот домен. Во [63], авторите го применуваат MapReduce моделот за вршење на анализи на податоци од високоенергетска физика и Kmeans кластерирање. Тие, исто така, имаат направено MapReduce имплементација базирана на стриминг и ја споредуваат нејзината ефикасност во однос на Hadoop. Нивниот заклучок е дека поголемиот дел од научните анализи кои имаат форма на SPMD алгоритам (една програма, повеќе податоци) може да имаат корист од моделот MapReduce и може да постигнат скалабилност и забрзување.

Во [57], авторите ја опишуваат имплементацијата на моделот MapReduce во Google. Имплементацијата е скалабилна и се процесираат терабајти податоци на илјадници машини. Исто така, скоро илјада MapReduce процеси се извршуваат на кластерите на Google секој ден. MapReduce моделот се користи за сортирање, рударење на податоци, машинско учење, генерирање на податоци на веб сервер, итн.

Во неговата теза [82], авторот предлага решение за симулација од молекуларната динамика базирано на Hadoop и MapReduce. Решението може да го предвиди времето на извршување на систем кој извршува симулација од областа на молекуларна динамика. Тој, исто така, го презентира и подобрувањето на перформансите и енергетската потрошувачката на решението кое е имплементирано во хибридна MapReduce средина.

Bunch и останатите [39] истражуваат кои научни пресметувачки проблеми може да се решат со користење на MapReduce, а кои не можат. Тие имплементираат различни нетривијални алгоритми користјќи MapReduce и ги мерат нивните перформанси. Заклучокот од нивното истражување е дека

MapReduce моделот не е погоден за итеративни алгоритми каде што во секоја итерација се извршуваат повеќе MapReduce процеси.

Во нивниот труд [197], авторите предлагаат архитектура за конфигурација имплементирана во научен прототип поставен на приватен облак и користат Hadoop за да постигнат скалабилност и толеранција на грешки. Експериментите ја покажуваат ефикасноста на предложениот модел. Во [218], авторите го опишуваат развојот на Hadoop-базирана научно пресметувачка апликација поставена на облак која обработува низи од микроскопски слики на живи клетки.

Додаток на Hadoop кој им овозможува на научниците да специфицираат логички прашалници над логички податочен модел е презентиран во [38]. Додатокот извршува прашалници како MapReduce програми дефинирани над логичкиот податочен модел. Целта на овој труд е да се намали вкупниот број на податочни трансфери, читања на далечина и непотребни читања.

3.3.3 Пресметување на просечни вибрациони потенцијали на осцилатори во околини на кондезирана материја користејќи Map Reduce и Hadoop

Теоретските модели во физичките науки, често се користат за да се разбере експериментално набљудуваното однесување на одредени физички системи или да се предвиди нивното однесување под одредени околности кои се релевантни за вистински или потенцијални технолошки примени на системите. Покрај добивање на пореален поглед на однесувањето на системот, теоретските модели може да бидат многу корисни за да се направи разграничување помеѓу различните фактори кои водат кон обсервација на одредени физички феномени или при пресметка на придонесот на различни фактори на одредени физички обсервации. Повеќето од експерименталните податоци, сепак, се собрани на конечни температури, обично температури над апсолутната нула.

Добар теоретски модел кој што има за цел да обезбеди реален опис на системот кој се симулира мора да ги земе в предвид динамичките ефекти во одредена временска рамка. Повеќето од моделите базирани врз основа на квантномеханички опис на физички систем со многу честички се базираат на истражувања на хипер рамнините на потенцијалната енергија (или одредени делови од овие површини), што значи дека тие не се во согласност со претходно наведените критериуми. За да се вклучи експлицитно динамичнкото однесување на проучуваниот квантен систем, истиот мора да се третира во рамките на квантната динамика. Сепак, целосниот и точен квантно динамички третман на системи со повеќе честички е премногу пресметковно скап. Но, таквиот целосен квантно-динамички третман е задолжителен само во одредени специфични случаи, обично кога фокусот на студијата се става на лесни честички (како на пр. водородни атоми).

Прифатлива алтернатива која се користи во литературата е прво да се изврши класично динамичка (или статистичко физичка, како на пример, Монте Карло) симулација на временска еволуција (или еволуција во имагинарно време) на системот, а потоа да се изберат разумно мал број на конфигурации (слики од класичната симулација) и да се вршат ригорозни квантномеханички

симулации само на овие конфигурации. Иако претходно споменатите динамички симулации се класични во ригорозна смисла, треба да се има в предвид дека потенцијалите за интеракција кои се користат за време на симулациите може да бидат изведени дури и од квантномеханички пресметки на високо ниво.

Во физичките науки, кога кондензираните системи (на пример, цврсти материи или течности) се моделираат со експлицитно вклучување на динамички ефекти, често се јавува следниот пресметковен проблем. Дадена особина на вграден атомарен/молекуларен систем во кондензирана фаза треба да се пресметува или со различни можни структурни конфигурации и понатаму да се бара просек на конфигурациите, или алтернативно, можно е да се генерира просечна конфигурација и понатаму да пресмета особината од интерес за таа конфигурација.

Проблемот за решавање на просечните вибрациони потенцијали на голем број на осцилатори во различни средини на кондензирана материја (примероци од статистичко физичка симулација) може да се стави во категоријата на проблеми со големи множества податоци. Притоа, потребно е да се примени дистрибуирано и паралелно процесирање на големото множество на податоци потребно за генерирање на просечниот вибрационен потенцијал. Еден од начините да се спроведе тоа ефикасно е со користење на програмскиот модел MapReduce и софтверската библиотека Hadoop. Главните причини за избор на софтверската библиотека Hadoop се следните: Има можност да обработува податоци паралелно; пресметковните решенија овозможени од Hadoop се скалабилни и флексибилни; дистрибуираниот датотечен систем овозможува брз пренос на податоци помеѓу јазлите; Hadoop е толерантен на грешки што значи дека ако еден јазол не функционира, задачата за извршување ќе се пренасочи кон друг јазол. Конкретно, целта на ова истражување е да се изврши ефикасна обработка на големи збирки на податоци кои се користат во научни апликации. Резултатите од истражувањето се објавени во [127].

На пример, доколку сме заинтересирани за анхармоничен осцилатор вграден во цврста или течна материја, вибрациониот потенцијал во следнава форма

$$V(r) = V_0 + 1/2k_2r^2 + k_3r^3 + k_4r^4 + k_5r^5 \tag{3.4}$$

може да биде пресметан со n конфигурации и потоа да се реши вибрационата Шредингерова равенка за секое $V_i(r)$. Во претходната равенка, r е соодветно избрана вибрациона координата, k_2 е константа на хармоничната сила, додека k_3, k_4 and k_5 се констатнти на анхармонична сила на степен 3, 4, и 5 соодветно [116] [115]. Во двата труда се користи овој пристап за генерирање на вибрационата густина на состојби за голем број на X-H осцилатори вградени во различни течни средини. Иако овој пристап е пресметковно изводлив, во некои случаи, особено ако сме заинтересирани само за просечната фреквенција (или поместување на фреквенција), пожелно е да се избегне експлицитна пресметка на вибрациониоте фреквенции со решавање на вибрационата Шредингерова равенка за сите $V_i(r)$. Наместо тоа, може да се користи една пресметка од овој тип, за просечна конфигурација или просечниот потенцијал во кондензирана фаза. Во ова истражување, се елаборираат претходните две идеи, земајќи го в предвид просечниот вибрационен потенцијал.

Алтернативно на просечната конфигурација, може да се генерира просечен вибрационен потенцијал во форма:

$$<V(r)>=<V_0> +1/2 <k_2> r^2+ <k_3> r^3+ \\ <k_4> r^4+ <k_5> r^5 \quad (3.5)$$

каде што е <> означува просек на временските конфигурации, а потоа се решава вибрациона Шредингерова равенка за таквата просечна функција на потенцијалната енергија. За да се илустрира овој концепт со одреден физички систем, предмет на внимание во ова истражување е флуороформ - диметилетер димер вграден во течен криптон. Главниот интерес за овој систем, што претходно е проучуван со спектроскопски техники, доаѓа од невообичаеното однесување на C-H вибрациониот режим на флуороформната група при соединување со диметилетерот. Се појавува сино поместување (наместо црвено поместување од „хемиска интуиција") на C-H фреквенцијата на истегнување.

Во тековното истражување, фокусот е ставен на развој на метод за извлекување на просечен X-H вибрационен потенцијал на истегнување со користење на пресметковната техника Map-Reduce. За таа цел се пресметуваат функциите на вибрациона потенцијална енергија за најмалку 50 C-H осцилатори на CF $_3$ H групата во рамките на CF $_3$ H - (CH $_3$) $_2$ O димерот на B3LYP, HF и MP2 нивоа на теорија. 6-31++G(d,p) базното множество се користи за орбиталното проширување во сите пресметки. Позициите на јагленородните и водородните во текот на „возбудувањето" на C-H вибрацијата на истегнување се генерираат со одредување на центарот на масата на C-H врската. Се користат мрежи од 20 точки за испитување на C-H функцијата на потенцијална енергија. Добиените податоци се интерполираат со полином од петти ред во C-H растојанието r (Равенка 3.4). Функциите од равенката 3.4 се трансформираат во Simons-Parr-Finlan координати $rho = r - r_e/r$ (каде што е r_e е вредноста на еквилибрум на C-H растојанието), и вибрационата Шредингерова равенка се решава со варијациски метод (линеарна варијанта). За таа цел, базните функции на хармоничниот осцилатор се користат како ортонормално базно множество. За генерирање на просечниот потенцијал од равенката 3.5 со Map-Reduce техника, се врши усреднување на вредностите на молекуларните потенцијални енергии (во Born-Oppenheimer смисла) за секоја r(C-H) вредност. Резултирачката функција за просечната вибрациона потенцијална енергија од равенката 3.5 се трансформира во SPF-тип на координати, а потоа вибрационата Шредингерова равенка се решава на варијациски начин. Имлементацијата на техниката Map-Reduce се прави во Hadoop, како што е објаснето подолу.

Сумарно, целта на нашиот алгоритам е да се пресметатат енергиите на просечниот вибрационен потенцијал за најмалку 50 C-H оцилатори на истегнување на CF_3H групата во CF_3H - $(CH_3)_2O$ димерот на B3LYP, HF и MP2 нивоа на теорија. Пресметувањето на просечните вредности, во нашиот случај енергиите на просечниот вибрационен потенцијал, е типичен Map-Reduce проблем. Секој документ кој го опишува истиот C-H осцилатор на истегнување во различна средина има две колони од кои едната ги содржи растојанијата r(C-H), а другата ги содржи вредностите на молекуларните потенцијални енергии (U).

Псевдо кодовите на методите Map и Reduce кои се користат во нашиот алгоритам се дадени во продолжение.

```
Method Map(String r, String U):
// r: input key r(C-H)
// U: input_value
for each r in all documents:
EmitIntermidiate(r,ParseDouble(U));

Method Reduce(String r, Iterator interm_vals):
// r: key, same as input_key
// interm_vals: intermediate values-
// list of all U-s group by r
double sum=0,result=0;
for each v in interm_vals:
sum += v;
result=sum/length(interm_vals);
Emit(AsString(result));
```

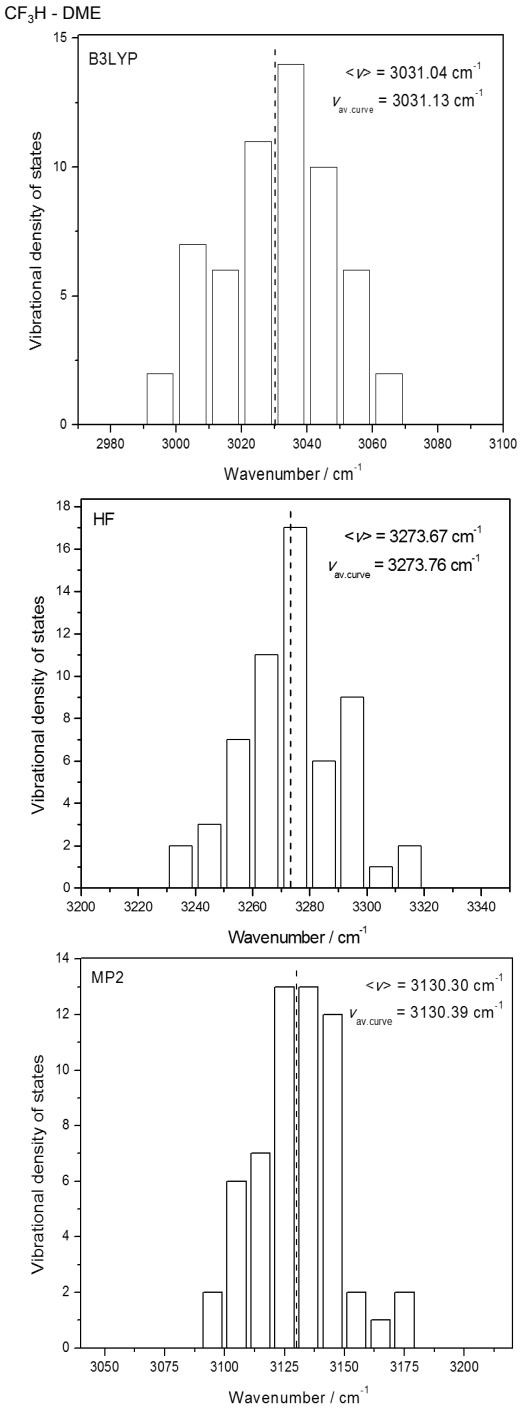
Алгоритмот е имплементиран во програмскиот јазик Јава и се извршува посебно на сите три нивоа на теорија (B3LYP, HF и MP2).

Главните резултати од ова истражување се сумирани на слика Сл. 3.7 a-c. На сликата Сл. 3.7 , се прикажани хистограмите на вибрационите состојби на густина (density-of-states (DOS)), генерирани од пресметаните вибрациони фреквенции од $|0> \rightarrow |1>$ C-H транзицијата на вибрационо истегнување заедно со делта функцијата (со испрекинати линии) која ја претставува фреквенција на $|0> \rightarrow |1>$ транзицијата добиена за просечен C-H потенцијал. На истата слика се прикажани и нумеричките вредности на соодветните фреквенции. Како што може да се забележи, доколку некој е заинтересиран само во просечните вибрациони фреквенции, а не за соодветните дистрибуции, овој пристап за просечен вибрационен потенцијал дава одлични резултати. Совпаѓањето меѓу просечните вибрациони фреквенции пресметани од DOS дистрибуциите, и вредностите на вибрационата фреквенција пресметани само од просечен вибрационен потенцијал е одлично, без оглед на нивото на теорија.

Резултатите покажуваат дека постои одлично совпаѓање помеѓу просечните вибрациони фреквенции пресметани од DOS дистрибуциите и вредностите на вибрационата фреквенција пресметани само од просечните вибрациони потенцијали. Со користење на резултатите, вибрационата Шредингерова равенка беше решена на варијациски начин. Бидејќи во научните домени постојат проблеми со обработка на големи податоци и барање на некакви средни конфигурации или вредности, парадигмата на MapReduce и Hadoop може да биде погодна за нивно решавање.

3.3.4 Откривање на магнетните својства на Al(III) јон во вода: хибридно статистички квантномеханички пристап со примена на Map-reduce техника

Темелното разбирање на карактеристиките на јонски/молекуларни видови во кондензирана фаза е од клучно значење за подобро разбирање и реално



Слика 3.7 – Хистограми на вибрационите состојби на густина (density-of-states (DOS)), генерирани од пресметаните вибрациони фреквенции заедно со функцијата делта

моделирање на процеси кои се одвиваат во биохемиски, геохемични и други средини. Сепак, развојот на сигурни и робусни теоретски модели кои експлицитно го вклучуваат влијанието на кондензираната фаза, особено нејзините динамички карактеристики е далеку од тривијална задача. Едната страна на

проблемот е да се најде суштински коректен начин да се вклучи кондензираната фаза во моделот, додека од друга страна да се земе в предвид пресметковниот аспект на проблемот. Поврзано со последниот проблем се јавува потребата за големи пресметковни ресурси, развој на нов ефикасен алгоритам за анализа на податоци, како и ефикасно користење на компјутерските ресурси (што повторно е проблем поврзан со програмирање).

Во ова истражување, се прави пресметка на магнетните својства на Al^{3+} јон во вода. Се разгледуваат основните аспекти во врска со развојот на пресметковен метод и пресметковните аспекти поврзани со ефикасноста на пресметковниот процес. Конкретно, се применува Map-Reduce пресметковната техника во одделните фази на пресметка во рамките на развиената методологија. Резултатите од ова истражување се објавени во [125] [123].

Al^{3+} во вода преставува релевантен систем во многу научни области меѓу кои и геохемија, биохемиски науки, како и хемија за животната средина. Постојат бројни истражувања за аспекти поврзани со хидратација на Al^{3+} во разредени и концентрирани водни раствори. Сепак, повеќето од методите кои имаат за цел експлицитно да го вклучат влијанието на кондензирана фаза врз својствата на јонот се засноваат или на кластер или на кластер + пристапи на поларизирачки континуум. Во првиот пристап (кластер) само најблискиот сосед во течна средина е експлицитно вклучен во моделот. Со други зборови, се испитуваат хипер рамнините на потенцијалната енергија на слободните кластери вклучувајќи го Al^{3+} јонот плус најблиските молекули на вода и се пресметуваат својствата на таквите кластери, со цел да се репродуцира однесувањето на системот во течна средина.

Се разбира, разгледувањето исклучиво на оние молекули од растворувачот во најблиското соседство на Al^{3+} јонот тешко може да доведе до задоволителни резултати ако некој е заинтересиран на комплетното влијание на растворувачот на овие својства. Добро е познато дека силно диполарна течност (како што е водата) врши значително електростатско влијание на високо наелектризирани јони како Al^{3+}. Поставување на јонот заедно со околината на најблиските соседи во рамките на остатокот од растворувач третиран како поларизирачки континуум, претставува многу пореална варијанта на моделот. На пример, третирањето на најголемиот дел од растворувачот како едноставен поларизирачки континуум не се занимава со специфичните нековалентни интермолекуларни интеракции помеѓу молекулите на водата, кои се наоѓаат во првата хидратациска обвивка (т.е. вистинските најблиски соседи на јонот) и оние кои се наоѓаат во рамките на втората обвивка. Таквите специфични интеракции, кои во овој случај се од типот на водородна врска може да доведат до повеќе значајни промени во електронската густина на молекулите на водата во втората обвивка, многу поголеми од оние предвидени со поедноставниот начин: кластер + пристапи на поларизирачки континуум.

За откривање на променливиот карактер на кондензираната фаза, прво се врши статистичко физичка Монте Карло симулација на Al^{3+} воден раствор проследена со квантномеханички пресметки на магнетните својства на наелектризирани вградени кластери со различна големина и комплексност. Во детали се разгледува соодветната застапеност на електростатското влијание во

раствор на овие својства користејќи пристап на просечна електростатска конфигурација на раствор (ASEC) кој што е многу поедноставен, бара помалку пресметковни ресурси и е широко распростанета методологија за просечна електростатска конфигурација на раствор. Во овој конкретен случај, ASEC пресметковниот метод се спроведува со користење Map-Eeduce техниката, која е исклучително погодна кога се во прашање вакви пресметки.

Целта на ова истражување е да се предложи модел што може да овозможи многу пореален опис на овој воден јонски систем, врз основа на хибридно статистичко физички квантномеханички пристап. Основната идеја на овој пристап е да се занимава исклучиво со динамичкиот карактер на течната средина, односно да не ги занемари термичките движења на молекулите во течна фаза. Откако ќе се генерира статистичко физики модел на течноста, тогаш се спроведува поригорозна квантномеханичка методологија за пресметување на магнетните својства на Al^{3+} јонот. Фокусот на истражувањето е насочен кон развој на метод за пресметување на магнетните својства на Al^{3+}, јон во вода, или попрецизно на $Al(H_2O)_6{}^{3+}$ видови во течна средина - $Al(H_2O)_6{}^{3+}(aq)$.

Овие хидрирани видови служат како стандард за споредба на магнетните својства на други можни водни Al(III) видови, односно како еден вид на внатрешен стандард во однос на кој се пресметуваат поместувањата. Од фундаментална и практична важност е да се развие робустен и сигурен метод за пресметување на магнетните својства на $Al(H_2O)_6{}^{3+}(aq)$, со соодветно вклучување на водна средина. Се разбира, една од можностите е пристап со конечни кластери, каде зголемувањето кластерите ќе се третира на статички или динамички начин. Од друга страна, многу поверојатен пристап би бил да се избере еден вид на просечна конфигурација генерирана од серија на статистичко физички симулации, за кои може да се врши пресметка на прилично високо ниво.

Целта на развиениот алгоритам е да се имплементира пресметковен метод за пресметување на просечната електростатска конфигурација на раствор (ASEC) користејќи ја Map-Reduce техниката.

Во опишаниот експеримент постојат вкупно 3000 конфигурации дадени во влезната датотека. Секоја конфигурација е дефинирана со 1 Al атом (опишан со x, y и z координати) и 3000 молекули на H_2O (секој H атом и O атом опишани со x, y и z координати). Целта е да се најде една просечна конфигурација која се состои од 1 Al атом и 3000 молекули на H_2O. Со цел да се комбинираат соодветните атоми (на пр. првиот кислороден атом од сите конфигурации), атомите се нумерираат со соодветни индекси.

Се дефинираат две дополнителни класи: Composite_key и Atom. Класата Composite_key го имплементира интерфејсот WritableComparable и ги преоптоварува трите методи: readFields(DataInput in), write(DataOutput out) и compareTo(compositekey o) [69]. Во класата Composite_key се декларирани две променливи: index и atom_name. Секој клуч од торката <клуч,вредност> е објект од класата Composite_key и секоја вредност е објект од класата Atom. Класата Atom го имплементира интерфејсот Writable и има три променливи од тип doulbe: x, y и z. Секој клуч или вредност кога се користи рамката Hadoop Map-Reduce го имплементира интерфејсот Writable (или интерфејсот

WritableComparable). Псевдо кодот на методите map и reduce кои се користат во нашиот алгоритам е даден во продолжение.

```
Method Map(LongWritable key, Text value):
 // key: input key  value: input_value
 Composite_key k;
 Atom v;
 for each line in value:
 {
       String [] array=line.split(" ");
       k=new Composite_key(array[1], array[2]);
       v=new Atom (array[3], array[4], array[5]);// x, y and z
       EmitIntermidiate(k,v)
 };

Method Reduce(Composite_key k, Iterator<Atom> interm_vals):
 // k: key  interm_vals: intermediate values -
 // list of all values(Atoms) grouped by k
 double sumx=0.0, sumy=0.0, sumz=0.0;
 Atom result;
 for each v in interm_vals:
 {
       sumx += v.x;
       sumy += v.y;
       sumz += v.z;
 }
 result.x=sumx/length(interm_vals);
 result.y=sumy/length(interm_vals);
 result.z=sumz/length(interm_vals);
 Emit(k, result);
```

За генерирање на структурата на водниот раствор од Al(III) јони се користи статистичко физички пристап реализиран со Монте Карло (MC) методот и алгоритмот на Метрополис. За таа цел се користи кодот DICE од статистичка механика [52]. Сите MC симулации се извршени на T = 298 K, P = 1 atm, користејќи ја експерименталната густина на вода во течна состојба 0.9966 g cm^{-3} за овие услови. Во секоја MC симулација, еден Al^{3+} јон е опкружен со 3000 молекули на вода во кубична кутија со должина на страната приближно 45 $\AA$, вклучувајќи ги и условите на периодични граници. Корекциите на долг опсег (LRC) на енергијата за интеракција се пресметуваат за парови на атоми кои се во интеракција и меѓу кои растојанието е поголемо од радиусот дефиниран како половина од должината на единичната ќелија. Придонесот на Lennard-Jones кон енергијата на интеракција над ова растојание се проценува под претпоставка дека има униформна дистрибуција на густината во течност (на пример g(r)$\approx$ 1), додека електростатскиот придонес се проценува со методот на реакција во поле вклучувајќи диполарни интеракции. Во сите MC симулации извршени во ова истражување, интермолекуларните интеракции

се опишани со сумата на Lennard-Jones 12-6 site-site енергии на интеракција заедно со условите на Coulomb:

$$U_{ab} = \sum_{i}^{a} \sum_{j}^{b} 4\varepsilon_{ij}[(\frac{\sigma_{ij}}{r_{ij}})^{12} - (\frac{\sigma_{ij}}{r_{ij}})^{6}] + \frac{q_i q_j}{4\pi\varepsilon_0 r_{ij}} \tag{3.6}$$

каде i и j се страните во интерактивните молекуларни системи a и b, а r_{ij} е растојанието меѓу атомите помеѓу страните i и j. Следната комбинација од правила се користи за да се генерираат двострани Lennard-Jones параметри ε_{ij} и σ_{ij} од едностранi параметри:

$$\varepsilon_{ij} = \sqrt{\varepsilon_i \varepsilon_j} \tag{3.7}$$

$$\sigma_{ij} = \sqrt{\sigma_i \sigma_j} \tag{3.8}$$

За вода, се користат потенцијалните параметри на SPC (simple point charge) моделот [28]. Заштитните магнетни тензори на јадрото се дефинираат како втор извод на енергијата (E) во однос на магнетниот моментот на X-тото јадро ($\overrightarrow{m}_x$) и надворешното магнетно поле ($\overrightarrow{B}$) [62]:

$$\sigma_x^{\alpha\beta} = \frac{\partial^2 E}{\partial B^\alpha \partial m_x^\beta}$$

каде што степените α и β го означуваат соодветниот вектор или компонентите на тензорот. ^{27}Al изотропните вредности на поместување се пресметуваат на следниот начин:

$$\sigma_{\text{iso}} = \frac{1}{3}(\sigma_{11} + \sigma_{22} + \sigma_{33})$$

За постигнување на калибрациона непроменливост (инваријанса) се користат два пристапи: метод на атомска орбитрала независна од калбирацијата GIAO (gauge independent atomic orbital) и метод на континуирнно множество на калибрациски трансформации CSGT (continuous set of gauge transformations). GIAO методологијата се базира на користење на бранови функции експлицитно зависни од полето [61, 212, 202]. Од друга страна, CSGT методот е базиран на изразот за компонентите од заштитиот тензор за X-то јадро во однос на индуцираната електронска густина од прв ред. Точни пресметки за последната количина се постигнуваат со вршење на трансформација на калибрација за секоја точка во просторот [110, 111, 22]. Сите квантномеханички пресметки се извршени со B3-LYP комбинациска функција на размена и корелација, како и со теоријата на петрубации од втор ред на Moller-Plesset (MP2), со користење на прилично големи 6-311++G(3df,3pd) базни множества за орбиталата експанзија.

Наједноставниот чисто електростатски модел на влијание врз магнетните својства на Al^{3+} јонот се состои од третирањето на сите молекули од растворувачот изграден од точкасти полнежи. Постојат неколку начини да се генерираат такви репрезентации на растворувач.

Една алтернатива е да се пресметаат магнетните својства на Al^{3+} јонот во различни „моментални" течни средини, земени како пресеци од MC симулации, во кои сите молекули на водата кои се наоѓаат во сферата со даден радиус на пример, еднаков на половина од должината на страната на единична клетка (па дури и поголем, земајќи ги в предвид периодичните гранични услови) се претставени со точкасти полнежи поставени на позициите на атомите на водород и кислород генерирани од страна на статистичко физичката симулација.

Другата алтернатива, која е имплементирана во ова истражување е просечна електростатска конфигурација на раствор (ASEC) развиена од Coutinho и останатите соработници [53]. Оваа алтернатива се состои од наметнување на атомични полнежи земени од M статистички некорелирани MC генерирани конфигурации, секоја скалирана со 1/M. Една конфигурација се состои од еден Al^{3+} ion опкружен со 100·x молекули на вода, каде што x е бројот на молекули на вода вклучен во конфигурацијата, избран така што растојанието O_{water}...Al е помало од граничната вредност на r_{tresh}.. Максималната вредност на r_{tresh}. е $a/2$, каде a е должината на страна од клетка добиена со MC симулациите. Полнежите на атомите на кислород и водород во молекулите на водниот раствор се еднакви на $q_O/100$ и $q_H/100$, каде q_O и q_H се соодветните полнежи на SPC моделот кои што се користеа за време на MC симулациите. Во овој контекст, се тестира конвергенцијата на пресметаната ^{27}Al изотропна константа на поместување со гранична вредност на растојанието и пресметковните методи искористени за постигнување на калибрациона непроменливост.

Табелата 3.2 дава преглед на добиените резултати. Како што може да се забележи, со такви големи базни множества како се користат во пресметките, GIAO и CSGT изотропните вредности на поместување конвергираат. Како за конвергенција со r_{tresh} вредност, може да се види дека резултатите се веќе конвергирани по вклучувањето на сите молекули на вода каде кислородните атоми се во сфера со радиус 5 $\AA$ околу централниот Al(III) јон. Бидејќи сите претходни вредности одлично конвергираат, дисперзијата на дистрибуцијата на соодветните вредности добиени од серијата пресметки за 100 различни средини на Al^{3+} катјон е мала, просечните вредности се во одлично совпаѓање со оние пресметани со ASEC пристапот. Пристапот ASEC е одлична апроксимација за да се добијат просечните изортопни вредности на заштитната обвивка на Al(III) јон во вода, потпирајќи се на квантномеханичката пресметка на само една конфигурација во течност (иако таквата конфигурација сигурно не е физичка).

Сепак, земајќи ги в предвид сите молекули од водниот раствор како точкасти полнежи во контекст на пресметување на магнетните својства на Al(III) во течност ова е прилично грубо приближување, особено ако треба да се добијат квантитативни сложувања со експерименталните податоци за некои други водени раствори што содржат Al. За таквите видови, изотропните вредности на поместување се изразуваат како поместувања во однос на водените Al^{3+} видови, и овој начин служи како еден вид на „внатрешен стандард". Ова особено важи во случај на оние молекули на вода кои се во најблиската околина на Al^{3+} јоните, односно во првата обвивка на хидратација.

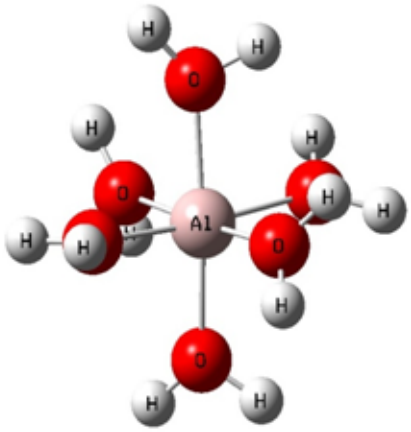
Интеракцијата на овие посебни молекули од растворувачот со централниот

Табела 3.2 – ^{37}Al изотропни вредности на поместување (изразени во ppm) пресметани на B3LYP/6-311++G(3df, 3pd) ниво на теорија, со GIAO и CSGT пристапи за се постигне калибрациона непроменливост, со различни r_{tresh} вредности (изразени во $\AA$).

	Слободен јон	5 $\AA$	10 $\AA$	15 $\AA$	20 $\AA$
GIAO	766.2468	766.2421	766.2421	766.2421	766.2421
CSGT	766.2477	766.2430	766.2430	766.2430	766.2430

јон е многу поинаква и многу посложена од вообичаена Coulombic интеракција како што се претпоставува со едноставен ASEC пристап. Додека ASEC пристапот е сигурно доволно добар за апроксимација на влијанието на најголемиот дел од останатите молекули од растворувач на магнетните својства на централниот Al^{3+} јон, молекулите на вода во првата обвивка треба да бидат третирани на поинаков начин, односно опишувајќи ги со целосна квантномеханичка бранова функција (наместо точкаст полнеж).

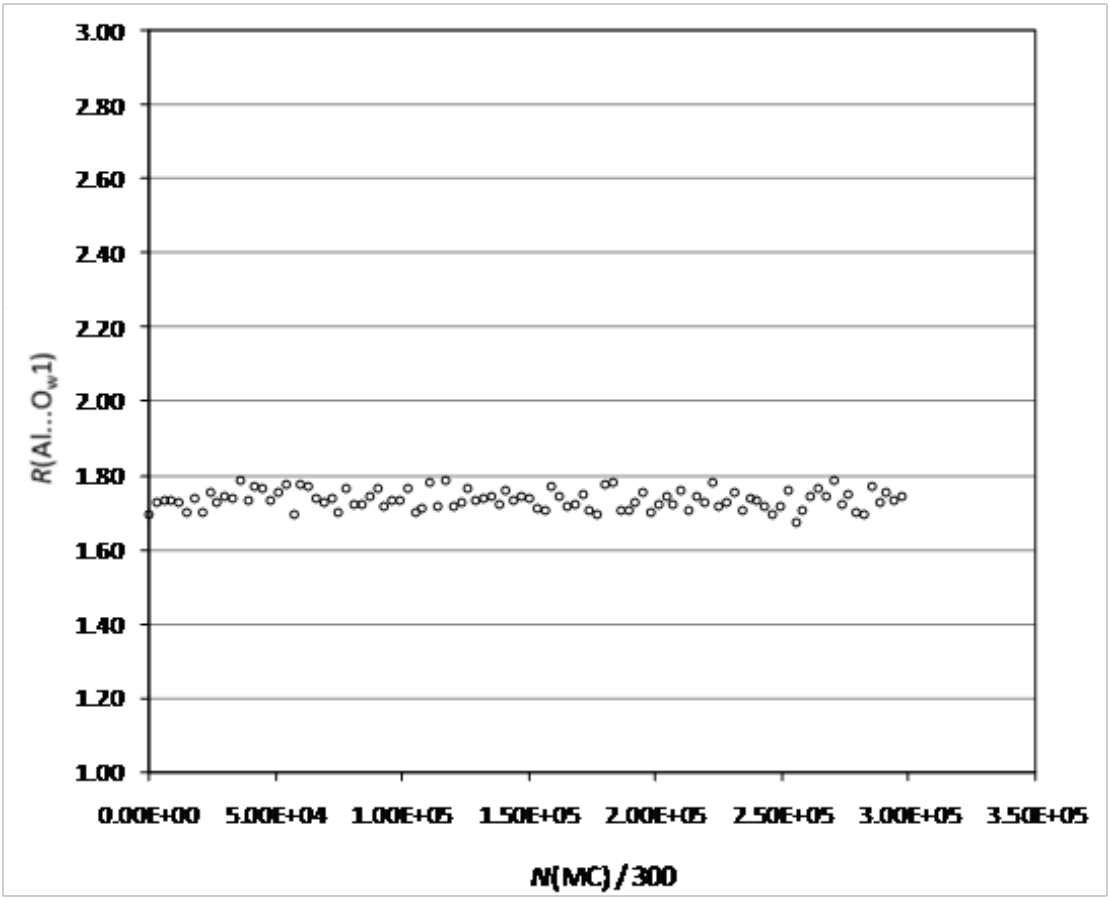
За да се демонстрира важноста на претходната изјава, извршени се тест пресметки на B3LYP/6-311++G(3df,3pd) CSGT and GIAO нивоа на теорија за $Al(H_2O)_6{}^{3+}$ видови вградени во молекули од растоворувачот, третирани како точкасти полнежи, земени од неколку пресеци на MC симулација која постигнала еквилибрум. Една типична конфигурација од овој тип е прикажана на слика 3.8.



Слика 3.8 – Типична конфигурација земена од MC симулација која постигнала еквилибрум, покажува поставеност на молекулите на вода во првата обвивка околу Al^{3+} јон.

Ваквите тест пресметки покажаа дека пресметаните изотропни вредности на поместување за централниот Al (III) вид значително се разликуваат од оние кои се добиени од страна на поедноставниот пристап ASEC. Додека конфигурација на останатиот дел на молекулите на растворувачот може безбедно да се третираат од страна на опишаниот метод ASEC, изборот на просечна „прва заштитна обвивка" е многу специфична и невообичаена пресметковна задача.

Еден начин е едноставно да се направи просек на Декартовите координати на секоја од молекулите на водата во првата обвивка за серија на MC генерирани конфигурации (после соодветна транформација на координатниот систем, доколку е потребно). Ваквиот пристап е оправдан бидејќи молекулите од првата обвивка на хидратација од растворот околу Al^{3+} јонот се прилично цврсто врзани, односно тие практично и немаат интеракција со молекулите на вода од втората обвивка во текот на симулацијата. Ова може јасно да се види од слика 3.9, каде што варијацијата на растојанието Al...O_w за една од молекулите на вода од првата обвивка околу Al^{3+} јонот е прикажана како функција од еден чекор на MC симулација. Сепак, овој пристап има одредени нејаснотии, кои треба да се решат на единствен начин. Корисна алтернатива би била за се направи усреднување на ориентационите параметри на секоја молекула на вода од првата обвивка околу Al(III), после соодветна транформација на координати за секоја MC генерирана конфигурација.



Слика 3.9 – Варијација на растојанието Al...O_w за една од молекулите на вода од првата обвивка околу Al^{3+} јонот како функција од еден чекор на MC симулација.

3.4 Дистрибуиран развој на научни апликации на грид

3.4.1 Карактеристики на дистибуиран развој и грид

Поголемиот дел од проблемите во различни научни области вообичаено се решаваат со поделба на проблемот на помали делови и со користење на дистрибуиран пристап на развој. Трендот на користење на овој пристап за развој расте паралелно со потребата да се постигне успешна соработка и координација на активностите во средина која овозможува висока скалабилност и пристап до хетерогени ресурси. Дистрибуираниот развој станува вообичаен пристап, бидејќи нуди современа околина за развој и овозможува координација и соработка меѓу учесниците во проекти кои се наоѓаат на различни места. Некои од предностите на дистрибуиран развој се: голем базен на ресурси, споделување на идеи и различни култури помеѓу програмерите, намалување на трошоци за развој, итн. [117].

Со оглед на брзиот развој на грид технологија и грид пресметувањето кое претставува чекор напред во дистрибуираното пресметување и обезбедува ефикасна инфраструктура за споделување на компјутери и податоци на динамички, нивната популарност расте брзо [135]. Грид посредник претставува слој за посредување меѓу мрежата и апликациите во грид инфраструктурата. Тој, исто така, управува со размена на информации, безбедност, и пристап. Развојот на софтвер кој ќе ја има улогата на грид посредник треба да ја земе в предвид комплексноста на развојот и управување со гридот [137].

Агенда за истражување за дистрибуиран развој на софтвер што се однесува само на неколку области, како што се тестирање, развој на софтверски алатки, управување со знаењето и прашања поврзани со процесот и метриките се презентирани во [184]. Стратегии за успешен дистрибуирани развој на грид во областа на физиката се прикажани во [134]. Во [98], авторите предлагаат нови парадигми за развој на дистрибуирани системи кои водат кон создавање на нови платформи за посредување.

Дистрибуираниот развој на софтвер е различен од традиционалниот централизиран развој на софтвер. Главната разлика помеѓу овие два пристапи на развој е просторната организација на развојните ресурси, вклучувајќи компјутери, луѓе, мрежни ресурси, итн.

Покрај добрата организација на ресурси, дистрибуираниот развој бара координиран процес на интеракција помеѓу членовите во проектот. Исто така, целта на проектот, големината на софтверските процеси и големината на тимот за развој треба да бидат земени в предвид [114].

Дистрибуирањето на софтвер може да се врши во неколку начини. Тоа е определено со географската локација и контрола на проектот. Подетално, софтверскиот проект може да се дистрибуира во истата земја каде што се наоѓа седиштето на развој или во друга земја. Контролата на проектот може да се организира на два начина: компанијата може да купи надворешен софтвер или може да ги понуди сопствените ресурси [185].

За ефикасно спроведување на дистрибуираниот процес на развој, потребно е да се воспостави ефикасна комуникација, соработка и контрола. Комуникацијата се дефинира како процес на размена на информации помеѓу учесниците

во софтверскиот проект; соработка преставува организација на работите и задачите за да се постигне посакуваната цел; контрола е процес на почитување на стандарди за квалитет, политики, формални принципи, итн. [136]. Концептот на грид а е посебен вид на концепт на дистрибуиран развој. Некои од главните карактеристики на грид [36] се прикажани во табела 3.3.

Табела 3.3 – Карактеристики на грид

Карактеристика	Опис
Скалабилност	Да може да се справи со милиони ресурси
Географска распределба	Мора да обезбеди добар концепт за распределба на ресурси на различни локации
Хетерогеност	Мора да поддржи различни видови на софтвер (датотеки, програми, податоци, итн) и хардверски ресурси (компјутери, мрежи, научни инструменти, итн.)
Споделување на ресурси	Различни организации мора да овозможат пристап до своите ресурси во мрежата.
Различни администрации	Секоја од организациите мора да дефинира специфични привилегии за пристап до нивните сопствени ресурси
Координација на ресурси	За да се обезбедат целосни пресметковни способности, мора да има координација на ресурсите во мрежата.
Транспарентен пристап	Контролата на пристап треба да овозможи целата мрежа да се гледа како еден виртуелен компјутер.
Сигурен пристап	Корисниците на мрежата мора да добијат високо ниво на перформанси, според утврдените барања за квалитет на сервис - QoS (Quality of Service).
Конзистентен пристап	За да се обезбеди скалабилност и сокривање на хетерогеноста на ресурсите, гридот да биде изграден со користење на стандардни протоколи и сервиси.
Постојан пристап	Гридот мора да обезбеди максимална ефикасност користејќи ги расположивите ресурси.

3.4.2 Проблеми поврзани со грид и со дистрибуиран развој

Еден од најголемите проблеми поврзани со концептот на грид е координацијата на ресурси кои обично се хетерогени и географски дистрибуирани. Покрај тоа, ресурсите може да припаѓаат на различни претпријатија кои немаат познавање едни со други. Ресурсите на гридот може да бидат управувани во различни административни домени кои имаат различен пристап и специфични политики за безбедност. Исто така, постојат и разлики во околините за развој како што се процесорски архитектури и оперативни системи [55]. Проблемите на дистрибуиран развој може да се поделат во неколку категории [30, 114, 136] прикажани во табелата 3.4:

Хетерогеноста на ресурси и динамиката може да резултираат во грешки и неуспеси. Транспортот на големи количини на податоци преку мрежа е ризичен и потребни се дополнителни напори за создавање на високо-пропусна

врска на долг период [55]. Имајќи ги в предвид овие барања, може да се заклучи дека инфраструктурата на гридот и околината на дистрибуираниот развој на софтвер се различни од традиционалните околини за развој на софтвер. За таа цел, за да се обезбеди поефикасен и поквалитетен процес на развој мора да се извршат дополнителни активности.

Доколу повеќето од овие проблеми се решени, инфраструктурата на гридот ќе обезбеди колаборативно инженерство, дистрибуирани пресметки, брз пренос на податоци и истражување на податоци [23].

Табела 3.4 – Проблеми на дистрибуиран развој

Проблем	Опис
Проблеми на култури	Разлики во национални (говорен јазик, вредности, практики), професионални, организациски, технички, а тимски култури. Овие проблеми се причина за отежната комуникација помеѓу учесниците во проектот.
Проблеми со комуникација	Промените во комплексните дистрибуирани инфраструктури и различните временски зони често водат до намалување на квалитетот на комуникација преку мрежата.
Проблеми во огранизација	Географски дисперзираните развивачи на софтвер мора да бидат ефикасно интегрирани во проектот. Тоа значи дека организацијата на процесот на развој и активностите за управување мора да се извршат навремено.
Проблеми со споделување на знаења	Споделувањето на знаења и искуства меѓу учесниците во проектот е важно за успешен развој бидејќи ги намалува трошоците и дополнителната работа (веб-базирани софтвери, како што е Wikis се корисни за напредок, следење и објавување на работата).
Проблеми со управување на процесот на развој	Промената на барањата и спецификациите, лошата организација и неформалната комуникација помеѓу членовите имаат негативно влијание врз процесот на развој и продуктивноста. Контролата на целиот систем станува се повеќе комплексна и тоа значи дека мора да има добар процес на управување со проектот.
Технички проблеми	Како резултат на дисперзираноста на локациите може да има недостаток на синхронизација на задачите. Пренос на податоци може да биде критичен поради бавни и несигурни мрежни врски.
Проблеми со управувања со ризици	Управувањето со ризици при дистрибуиран процес на развој е отежнат поради честа појава на дефекти. Потребни се дополнителни напори за управување со ризик и контрола.
Проблеми со следење на грешки и промени	Следењето на грешки и промени е тежок процес поради дистрибуираната околина. Тоа значи дека потребата за систем за следење на квалитет е од суштинско значење.

Проблеми со поседување	Може да не биде јасно чија е одговорноста кога некој од ентитетите во мрежата нема да успее. Ова може да се случи, бидејќи понекогаш луѓето од различни организации се одговорни за развој на истиот ентитет. Ентитетот треба да се измени и тестира повторно.

3.4.3 Развој на грид посредник

Карактеристики на грид посредникот

Посредникот (middleware) е софтвер кој обезбедува хомогена инфраструктура и единствен пристап до сите ресурси во хетерогени и географски дистрибуирани грид околини [98].

Имајќи ја в предвид комплексноста на целокупната организација и разновидноста на субјектите кои се вклучени во процесот на развој, потребата за управување на активностите за развој и ресурси е од суштинско значење. Така, главната идеја зад развојот на софтверот за грид посредник е да се овозможи конзистентна и ефикасна пресметувачка околина.

Задачите на посредникот можат да бидат поделени во пет категории: извршување задачи, безбедност, пренос на датотеки, информации и управување со датотеки [137]. Грид посредникот е дел од типичната грид архитектура. Архитектурата на гридот се состои од четири слоеви: хардверски слој (компјутери, уреди за складирање, мрежи, итн); јадро (ко-распределба на ресурсите, далечинско управување со процеси, безбедност и QoS); корисничко ниво (високо ниво на апстракција обезбедено од страна на околини за развој и програмски алатки, распоред на задачи); апликациско ниво (апликации и портали) [20].

Проблеми при развој

Развојот на софтвер за грид посредник е предизвикувачки проблем за софтверските инженери, бидејќи софтверот треба да може да се споделува, повторно да се користи, и неговиот развој да биде продолжен од други програмери во иднина [137].

Првенствено, проблемите во процесот на развој на посредникот се ориентирани кон можност за пренос и основна функционалност. Овие проблеми може да резултираат со неуспешно извршување на задачи, исто така, и во контролирани грид средини [96].

Дополнителен предизвик за програмерите е приспособливост на системот и барањата за квалитет [75]. Проблемите можат да настанат како резултат на неспроведување на стандардите за квалитет, особено кога гридот е наменет за јавна употреба.

Други проблеми при развојот на посредникот се способноста да се провери валидноста на изворниот код и можноста да се справи со грешки [48]. Проверката на валидноста на изворниот код побарува добар процес на тестирање, но тоа не е лесно да се направи, бидејќи посредникот мора да биде развиен за поддршка на хетерогени и комплексни средини.

Обезбедувањето на интероперабилен систем и автономни услуги може да предизвика голем број на неочекувани грешки бидејќи тоа зависи од повеќе фактори како што се: овозможување на различни сценарија за пренос во постоечката грид инфраструктура, можност за користење на услуги како самостојни ентитети во различен контекст, итн. [138].

Развојот на софтвер за грид посредник софтвер бара познавање на: грид инфраструктурата, услугите кои ги нуди, користените технологии за развој, стандардите кои мора да бидат земени в предвид при кодирање, тестирање, итн. Покрај тоа, програмерите треба да се грижат за расположливите ресурси и перформансите на системот.

3.4.4 Подобрување на квалитетот на дистрибуиран развој

Откривањето на недостатоците при процесот на дистрибуиран развој на софтвер, во споредба со централизиран софтвер, доведува до заклучок дека развојот на дистрибуиран софтвер бара подобрување на дополнителни активности како што се: комуникација, соработка, планирање, организирање, безбедност, тестирање, итн. Голем дел од проблемите за време на дистрибуиран развој или развој на грид посредник се јавуваат како резултат на ненавремена и лоша организација, без соодветни планови за тестирање, како и недоволна документација. Резултатите од ова истражување се објавени во [121].

За да се подобри квалитетот на дистрибуираниот развој на софтвер, треба да се вклучат одредени практики од софтверското инженерство кои решаваат дел од проблемите опишани во претходните секции. Практиките кои треба да се вклучат може да се поделат во три категории: организација, развој и тестирање.

Делот на организација се состои од неколку активности: организација на расположливите ресурси, креирање на план за развој и активности за подобрување на комуникацијата и колаборацијата на членовите вклучени во софтверскиот развој. Ова е важен чекор во дистрибуирана околина, првенствено поради географски дистрибуираните ресурси, како и големината и комплексноста на проектот. Еден начин да се подобри организацијата во процесот на развој е да се зголеми комуникацијата и соработката помеѓу учесниците. Видео конференциите и онлајн интеракциите се погодни за воспоставување успешна комуникација и соработка. Изготвувањето на план за активност треба да се базира на стандарди кои се општо прифатени како и на оние кои ќе се воспостават интерно во рамките на проектот.

Развојот на софтвер во дистрибуирана околина има потреба од зголемена интеракција помеѓу програмерите, документирање на секој дел од процесот на развој и интеграција на софтверските компоненти. Интеграцијата на делови од софтверот е процес кој бара правилен развој на секој од деловите поединечно. Неопходно е да се направат шаблони за документи, особено оние во кои се опишуваат промени на изворниот код и софтверските барања. По развојот на секоја компонента, мора да има соодветна документација за инсталација и употреба. И покрај ова, расположливите ресурси мора да се користат ефикасно и оптимално. Ограничувањата на ресурсите и нефункционалните барања мора

да бидат земени во предвид. Грид посредникот е најкритичниот дел во развојот каде што се развиваат основните сервиси за поврзување на апликациите и хардверските ресурси.

Потребно е да се дефинира план за тестирање и да се доставуваат тест извештаи после секој извршен тест. Тестирањето на дистрибуирани софтверски решенија бара дополнителни активности при валидација. Тестовите треба да се дефинираат пред да биде напишан изворниот код. Покрај тоа, мора да се применува тестирање на помали целини од кодот и притоа да се внимава сите барања да бидат задоволени. Неопходно е да се изврши интеграциско тестирање кое ќе ја провери валидноста на системот како целина. По секоја направена промена на изворниот код мора да се примени регресиско тестирање со кое ќе се утврди дека промената не го загрозила правилното извршување на останатите функции на софтверот. Тестирањето на грид посредникот на мора да се прави често и во согласност со планот за тестирање во рамките на проектот.

Итеративниот развој е добро решение за успешен дистрибуиран процес на развој. Квалитетот на дистрибуираниот развој на софтвер зависи од фактори кои директно или индиректно учествуваат во процесот на развој. Покрај стандардите за квалитет на развој на софтвер, искуството за развој на ова поле е од суштинско значење.

3.5 Регресиско тестирање на научни апликации

Напредокот во истражувањето во научните области постојано ја наметнува потребата за создавање на научни апликации. Една од дефинициите за научни апликации ја опишува научната апликација како математички модел кој симулира природен феномен[222]. Научните апликации преку создавање на математички модели можат да вршат симулација на многу сложени проблеми од реалниот свет кои можат исто така да се користат и за идни истражувања.

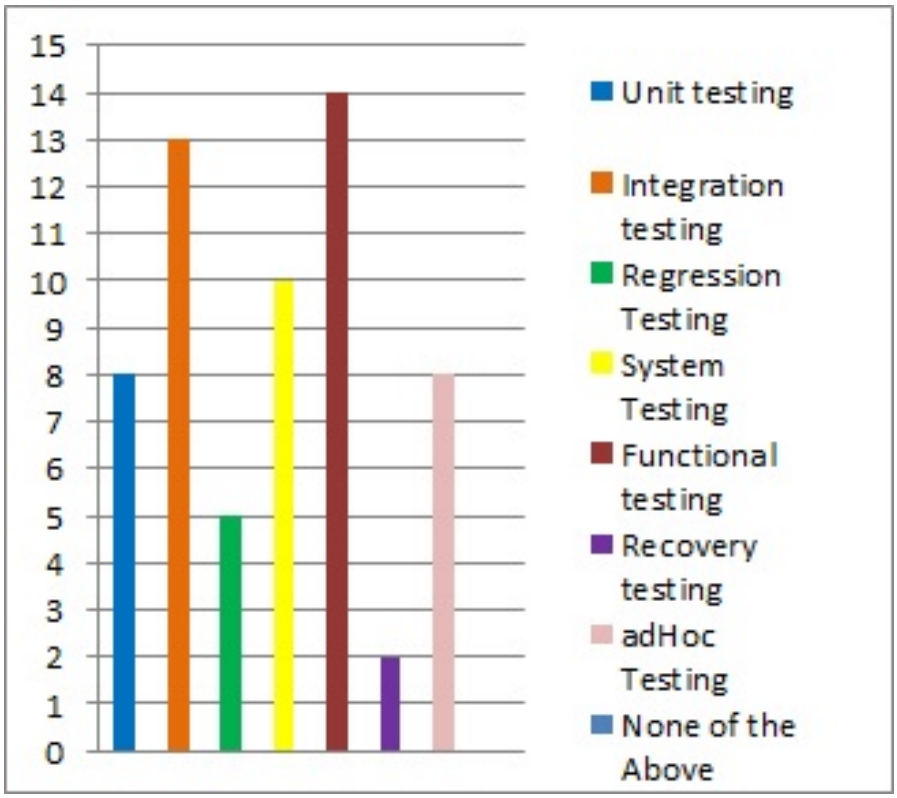
Потребата за создавање на големи и комплексни математички модели и математички пресметки ја зголемува веројатноста за грешки, а со тоа се зголемува потребата за вршење на промени во различни делови од апликацијата. Со успешното тестирање по извршената промена се потврдува дека системот е во стабилна состојба и дека не загубил некоја функционалност.

Земајќи го в предвид фактот дека промените се составен дел на процесот за развој на научните апликации, регресиското тестирање е од суштинско значење поради постојаните промени во барањата [139]. На пример, резултатите добиени од физички експерименти може да бидат причина за промени во барањата и изворниот код на апликацијата.

Главниот мотив за ова истражување доаѓа од резултатите добиени од анкета спроведена меѓу 20 научници во HP-SEE [122] (High-Performance Computing Infrastructure for South East Europe's Research Communities) проектот. Резултатите од ова истражување се објавени во [126]. Целта на ова истражување е да се подобри процесот на тестирањето на научни апликации, особено регресиското тестирање. Главниот придонес е тоа што се прави адаптација на некои постоечки практики на софтверско инженерство кои се користат за развој на комерцијални софтвери, каде што постојат клиенти. Разликата помеѓу

тестирањето на научниот и комерцијалниот софтвер е во методите за тестирање. На пример, една од клучните разлики е во дефиницијата на барањата и неможноста двата вида на софтвер да се тестираат на ист начин. Со оглед дека кај научниот софтвер не постојат клиенти, проверката на точноста на научниот софтвер е тешка задача. Вообичаено, за да се провери дали софтверот е коректен потребно да се изврши вистински експеримент [182].

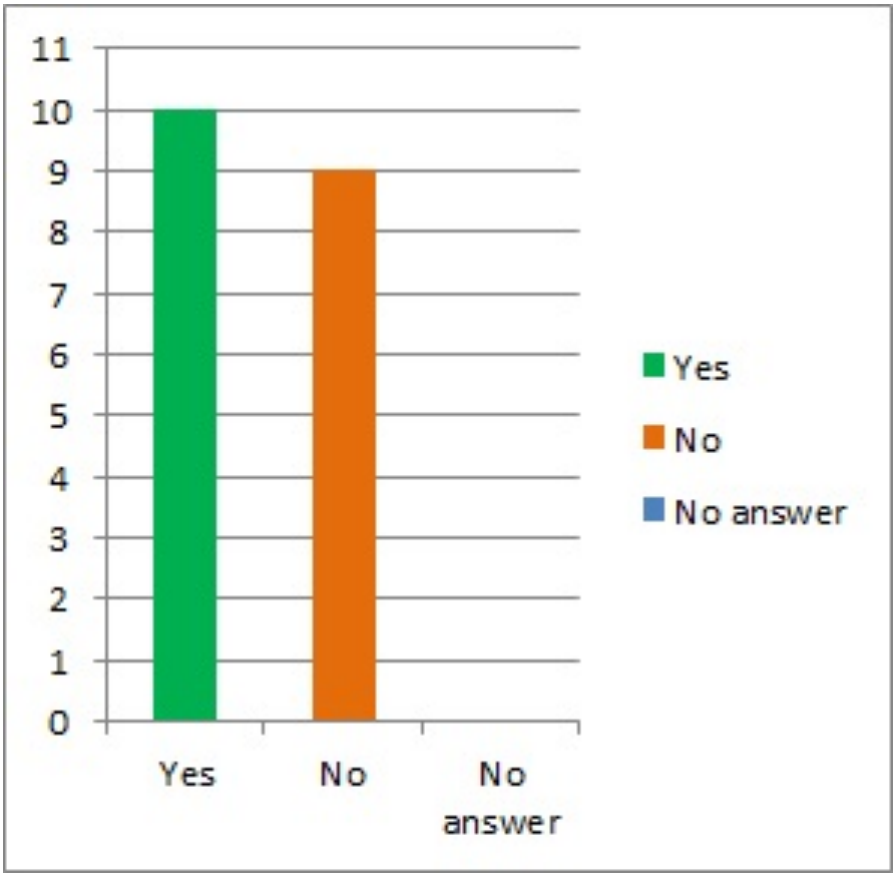
Со цел да се покаже практиката на научниците во проектот, резултатите на прашањето кои видови на тестирање научниците применуваат во процесот на развој на апликации се прикажани на слика 3.10. Тие покажуваат дека повеќето научници применуваат интеграциско и функционално тестирање, но само пет научни тимови применуваат регресиско тестирање, а само два научни тимови применуваат тестирање за обновување. Слика 3.11 ги презентира резултатите на прашањето за поврзувањето на барањата и тестовите. Најголем дел од научниците не ги дефинираат барањата или пак ги дефинираат во несоодветна форма.



Слика 3.10 – Одговори на прашањето: „Кои видови на тестирање се применуваат во вашите проекти?"

3.5.1 Интервју

Со цел да се добијат порелевантни резултати, направено е интервју со научник (хемичар) кој има големо искуство во развивање на научни апликации и еден од учесниците во HP-SEE проектот. Интервјуто содржи вкупно 12 прашања, кои главно се однесуваат на промените во барањата, тест случаите



Слика 3.11 – Одговори на прашањето: „Дали ги поврзувате барањата со соодветните тестови?".

и регресиското тестирање. Кратката верзија од интервјуто е дадена во продолжение. За секое прашање (П) приложен е одговорот (О) и дискусија (Д).

П1: Колку често се случуваат промени во барањата при развој на научни апликации апликации?

О1: Постојат две можни причини за промени во барањата во текот на развојот на еден софтвер. Првата се однесува на новите научни прашања кои би можеле да се појават во текот на истражувањето. Втората, од друга страна, доаѓа во игра кога истражувачот смислува нов пристап кој е подобар во однос на оној што веќе се користи до тој момент. Спроведувањето на промените често не бара значителни промени во сите сегменти на кодот, туку само во рамките на ограничен дел од него. Исто така, втората причина може да се однесува на моделирање или опишување на одреден физички систем на посоодветен начин.

Д1: Напредокот во истражувањето и новите резултати добиени од експериментите се најчестите причини кои доведуваат до промени во научните апликации. Научниците знаат дека можноста за такви промени е голема. Како резултат на тоа, потребно е да се прават промени во неколку сегменти на кодот, но тоа да не доведува до уништување на останатите делови кои функционираат коректно. Исто така, потребно е да се задржи концептот на функционирање на системот во целина. Научниците обично користат веќе

развиени изворни кодови за да го решат проблемот во нивната област на истражување и се обидуваат да направат соодветни измени за да се прилагодат на специфичните потреби за решавањето на проблемот. Најчесто овие кодови се развиени од страна на други научници во истражувачката заедница. Научниците се соочуваат со проблеми кога треба да се прават промени на овие кодови, бидејќи обично тие не се соодветно документирани.

П2: Како промените во барањата влијаат на кодот? Дали предизвикуваат големи промени во изворниот код?

О2: Тоа всушност зависи од причините за промените во барањата. Ако е потребна само мала промена во алгоритмот или додавање на уште една променлива која треба да се пресмета, често се потребни мали измени во изворниот код. Напротив, кога има за цел да се вклучи во истражувањето, новиот алгоритам или пристап, често се потребни повеќе промени на изворниот код, а во одредени случаи, многу е полесно да се напише кодот од почеток, отколку да воведат промени во тековната верзија. Ова прашање, исто така, значително зависи од програмерските вештини на истражувачот. Ако програмерот има напредни програмерски вештини, кодот може да биде толку флексибилен што се потребни само мали промени во изворниот код за широк спектар на промени во функционалноста на кодот. Ова, за жал, не се случува секогаш.

Д2: Проблемите со кои се соочуваат научниците кога треба да се направат големи промени во алгоритмот, најчесто се јавуваат поради несоодветна документација и слаба прегледност на изворниот код. Понекогаш, кодот кој бил развиен пред одреден временски период не може да се протолкува лесно. Затоа, за нив полесно е да развијат нов алгоритам за решавање на истиот проблем. Научниците ретко ја земаат в предвид комплексноста и јасноста на кодот, бидејќи најважното нешто за нив е точноста на резултатите кои алгоритмот ги дава. Ова се случува бидејќи научниците ги учат програмските вештини самостојно или од други научници и ретко ги применуваат практиките за софтверското инженерство.

П3: Кога се случуваат најголемиот број на промени (во која фаза на развојот)?

О3: Откако кодот е тестиран на широк спектар на системи. Во оваа фаза, често, повеќето од грешките се идентификуваат (концептуални и тривијални). Меѓутоа, исто така, многу промени може да се случат во било која фаза кога доаѓа до промена во барањата.

Д3: Резултатите од анкетата спроведена меѓу научниците во проектот HP-SEE покажа дека повеќето од тимовите за развој не се сигурни дали барањето содржи грешки. Ова се случува поради недостаток на: генерирани тест случаи, чести тестирања, документација и примена на практиките за софтверско инженерство. Вообичаено научниците вршат тестирање по одреден период на време кога веќе има многу линии код. Овој пристап предизвикува грешките да бидат откриени премногу доцна, а со тоа се зголемува бројот на промените што треба да се направат.

П4: Колку време е потребно да се направи промена во изворниот код после секоја промена во функционалноста?

О4: Тоа зависи од сложеноста на корекцијата или новата функционалност. Во повеќето случаи, особено ако се потребни само мали промени, измените во

изворниот код се брзи. Во некои случаи каде што има концептуални промени, промената на изворниот код може да одземе значително време.

Д4: Промените кои се јавуваат како резултат на додавање на нови функционалности во кодот или промена на концептуалениот модел обично одземаат многу време. Тоа особено е случај ако кодот е лошо напишан и без документација.

П5: Дали применувате регресиско тестирање? Ако применувате, како?

О5: Најчесто да, освен ако промената е исклучително мала или прилично тривијална. Вообичаено повторувам одредени калкулации на системски примероци или некои прототип системи (на пример, аналитичко-решливи системи).

Д5: Научниците обично извршуваат тестирање по промените, но не секогаш навреме. На пример, може да се направат неколку промени во изворниот код, а потоа тоа кога се врши тестирање да се пронајде грешка. Во овој случај, тешко е да се утврди причината за нејзиното настанување. Регресиско тестирање мора да се применува во апликации каде што промените се случуваат многу често. Регресиското тестирање одлучува дали новата верзија на апликацијата е стабилна. Соодветна документација за кодот кој ја опишува новата промена во функционалноста е исто така многу важен фактор, бидејќи не смее да се дозволи да се прават промени во кодот, а да се задржи старата документација.

П6: Дали избегнувате регресиско тестирање по промените и зошто?

О6: Јас обично не инсистирам на избегнување на регресиското тестирање (освен ако промената е навистина тривијална, на пример, подобрување на вредноста на основните или други константни).

Д6: Регресиското тестирање се користи за да се потврди функционирањето на апликацијата по промената. Оваа метода на тестирање обично се користи од страна на научниците, но промените не се документирани, а претходната верзија на апликацијата не се чува. Понекогаш тестирањето по голема промена може да даде точни резултати, но подоцна може да биде потребно да се врати претходната стабилна верзија на апликацијата.

П7: Дали правите промена на тест случај по промената на барањата на апликацијата кои се поврзани со тест случаите? Ако не, зошто?

О7: Промени во тест случаите се потребни ако постојат значителни промени во функционалноста на кодот, на пример, ако кодот е применлив за друга група на системи, или ако се воведат некои нови концепти. Ако промените во кодот вклучуваат само можност да се пресметаат дополнителни системски особини, тогаш вообичаено не е потребно да се прави промена на тест случаите.

Д7: Тест случаите за научните апликации вообичаено се напишани во слободна форма, без користење на стандардизирани шаблони за тестирање. Згора на тоа, промените кои се направени во барањата не секогаш се одразуваат во тест случаите. Научниците обично не создаваат тест случаи за секое барање, бидејќи не пишуваат соодветна спецификација на барањата.

П8: Како создавате тестови? Или, доколку не создавате тестови, како знаете кои делови од апликацијата се тестирани?

О8: Тестовите вообичаено се едноставни прототип системи, за кои потребните параметри се или познати или може да се пресметаат со, на пример, аналитички пристап.

Д8: Со оглед на фактот дека научните апликации претставуваат симулации на проблеми во реалниот свет, честопати не постојат точни решенија кои може да се предвидат однапред, туку се бара приближно решение. Одговорот покажува дека тестовите обично се изведува со помош на системи кои имаат веќе познати аналитички решенија. Овој пристап го отежнува процесот на спецификација на тест случаи. Сепак, треба да се создадат тест случаи кои ги покриваат барањата на апликацијата.

П9: Дали несоодветното тестирање негативно влијае врз процесот на развој на апликациите?

О9: Секако, тоа може да ги опфати и концептот на кодот и неговата функционалност.

Д9: Научниците се свесни дека со тестирање на погрешен начин и недоволно тестирање може да се наруши функционалноста на апликацијата. Подобрување на начинот на тестирање на апликациите за голема мера може да го зголеми нивниот квалитет.

П10: Користите ли некои алатки во процесот на тестирање и доколку користите, за кои алатки станува збор?

О10: Не.

Д10: Постојат многу алатки со кои може да се автоматизира процесот на тестирање и исто така, да се обезбеди поширок спектар на опции за тестирање. Има алатки кои создаваат тест случаи, извршуваат тестови, водат евиденција на тест случаи и автоматизираат многу од работата што се врши рачно. Недостатокот на користење на алатки за тестирање во текот на процесот на развој го зголемува времето потребно за тестирање и развој на апликации.

П11: Дали ги евидентирате сите промени во барањата и изворниот код? Имате ли некоја документација за тоа? Користите ли некои шаблони?

О11: Јас само додавам коментари во текот на изворниот код. Вообичаено, не подготвувам документација за високо специјализираните кодови. Меѓутоа, ако се планира да се направи пофлексибилен код, за една поопшта цел, тогаш вистинската употреба на кодот многу ќе зависи од квалитетот на документацијата.

Д11: Една од главните карактеристики на научните апликации е недостатокот од документација. Научниците вообичаено не создаваат документација за процесот на развој на софтвер. Тие најчесто ги дефинираат барањата и тест случаите на неформален начин. Генерирањето на документи е една од основните работи кои можат да го зголемат квалитетот на процесот на развој на апликацијата. Секако, главната цел на процесот на развој не е создавање на документи, но тие служат како дополнителен механизам за поддршка и за справување со промените во барањата и изворниот код. Постојат голем број на шаблони понудени од страна на компаниите кои се занимаваат со софтверско инженерство, кои со мали измени ќе се вклопуваат во развојот на научни апликации.

П12: Мислите ли дека регресиското тестирање е важно?

О12: Секако дека е. Тоа може да помогне и во временската линија на развој на апликациите, како и во нејзината коректност.

Д12: Регресиското тестирање е еден од клучните фактори за подобрување на квалитетот на научните апликации. Овој тип на тестирање е особено

важно за апликации каде што честопати се случуваат промени во барањата и изворниот код.

3.5.2 Практики за подобрување на регресиско тестирање

Пред почетокот на процесот на развој, научниците треба да ги дефинираат барањата на апликацијата. Сите барања не може да се предвидат однапред, бидејќи тоа зависи од напредокот во истражувањето. Исто така, промените во барањата може да се случуваат многу често. Се препорачува итеративен развој на научни апликации за да може да се додаваат функционалности во секое повторување[102]. Неопходно е да се утврдат очекуваните резултати за секоја функционалност. Ако не е можно да се предвиди точната вредност, тогаш потребно е да се дефинира опсегот на вредности каде што припаѓа решението.

Процесот на дефинирање на барањата помага за спецификациите на тест случаите и во процесот на тестирање на апликацијата. Добро одредените барања ќе доведат до попрецизно дефинирање на тест случаите и подобра организација на целокупниот процес на развој.

Тест случаите може да бидат генерирани автоматски со користење на статички пристап кој генерира влезови од некој вид на модел на системот (тестирање базирано на модел) и динамички пристап кој генерира тест случаи преку извршување на програмата [19].

Истражувањето покажа дека научниците не употребуваат никакви дополнителни алатки за автоматизирање на процесот на тестирање и споредување на резултатите со очекуваните резултати. Ова е особено важно кога се извршува регресиско тестирање бидејќи се повторуваат исти тест случаи.

Изборот на тест случаи пред да започне процесот на регресиско тестирање е еден од најважните предуслови за подобрување на квалитетот на апликациите. Одлуката за тоа кои тест случаи треба да бидат избрани и повторно да се извршат по промената зависи од многу фактори. Во изборот на тест случаи кои треба да се извршат повторно, вклучени се и оние тест случаи кои се поврзани со направените промени. Со добра техника при изборот на тест случаите се елиминираат непотребните тест случаи и се дава приоритет на случаи си повисока веројатност за откривање грешка.

Процесот на извршување на релевантните тест случаи може да се подели на три категории: Тест случаи поврзани со промената во барањата; Тест случаи поврзани со изворниот код кој е променет; Извршување на тест случајот што е променет. Втората е опција е најтешка за извршување – да се пронајдат тест случаите каде што има промена во изворниот код. За полесно препознавање на тест случаите кои се однесуваат на одредена промена во кодот, потребно е кодот да се организира во помали независни единици – функции. За добра техника при избор на тест случаи потребна е соодветна спецификација на барањата и тест случаите, како и максимална прегледност и организацијата на деловите од изворниот код. Исто така, ако други тест случаи зависат од тест случај кој се менува, потребно е да се извршат сите тест случаи кои зависат од него.

Дијаграмот на активностите на чекори пред и по процесот на регресиско тестирање, кога треба да се додаде нова функционалност или треба да се промени постоечка е прикажан на слика 3.12. По менување на барањето, кога се додава нова функционалност или се менува постоечка, прво треба да се направи корекција на тест случаите. Тоа значи дека сите тест случаи опфатени од промената на барањето мора да бидат изменети. Исто така, соодветни измени треба да се направат и во изворниот код.

Една од најважните работи е изборот на множество на тест случаи за регресиско тестирање. Во овој чекор треба да се опфатат и променетите тест случаи. Потоа треба да се изврши регресиското тестирање.

Сликата 3.13 ги претставува чекорите пред и по регресиското тестирање кога е пронајдена грешка во апликацијата. Откако ќе се пронајде грешка, треба да се идентификуваат причините и да се направат соодветни промени во изворниот код. Пред да се направи тестирањето, повторно треба да се направи селекција на тест случаи. Доколку некој од тест случаите биде неуспешен, повторно треба да се повторат сите чекори.

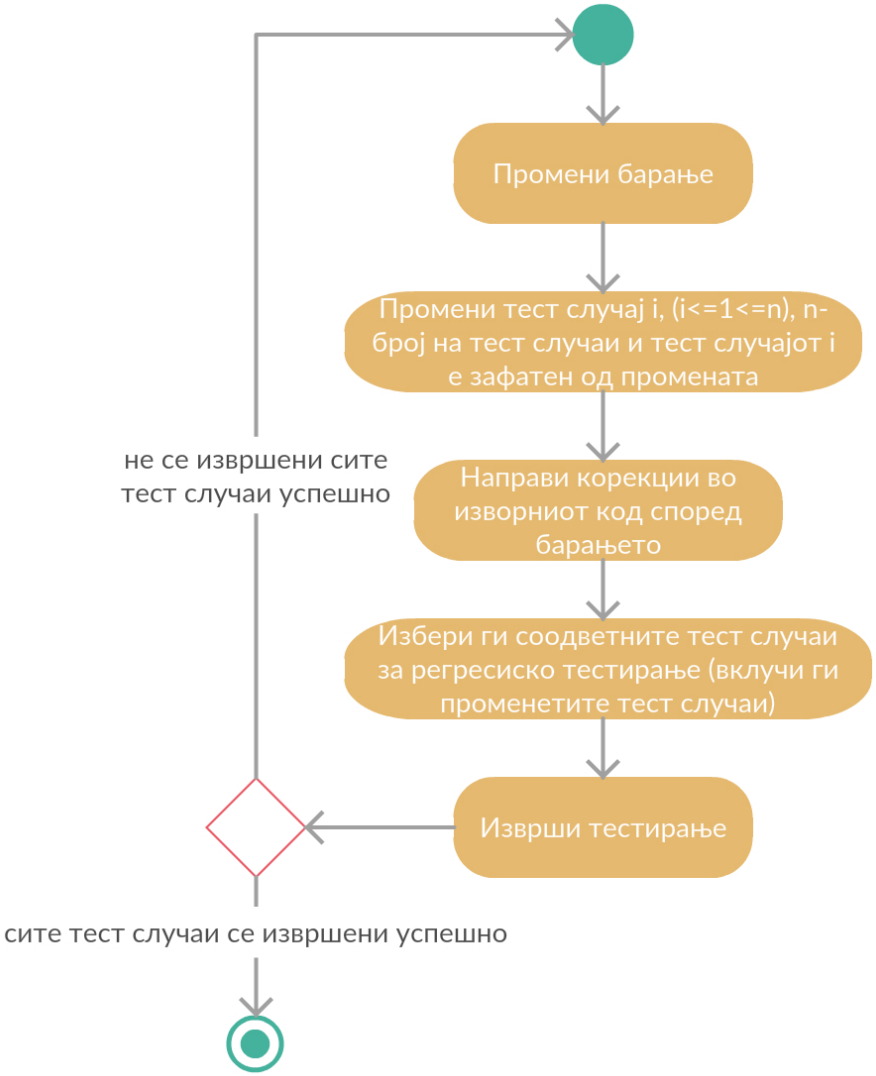
Регресиското тестирање е многу важен чекор во процесот на развој на научни апликации. Реализацијата на регресиското тестирање најмногу зависи од спецификацијата на барања и тест случаите. Кога при развојот на научни апликации не е можно да се дефинираат сите барања на почетокот, добра практика е процесот на тестирање да се одвива во фази (повторувања). Покрај тоа, во секое повторување треба да се тестираат барањата кои биле додадени или променети. Овој тип на тестирање помага во намалувањето на бројот на грешки кои се случиле како резултат на промена на барање или промена во изворниот код на научната апликација.

3.6 Тестирање при агилен развој на софтвер

Тестирањето на софтвер не треба да се дефинира само како процес за пронаоѓање на грешки во работата на софтверот, туку како ментална дисциплина каде што тестерите можат да ја покажат својата креативност, да најдат најприфатливи методи за тестирање или да предложат нови идеи за тестирање. Развиените компании имаат напредни вештини и техники за тестирање и го сметаат тестирањето како психичка дисциплина која може да помогне во сите процеси во животниот циклус на софтверот [15]. Досегашните истражувања наведени во дисертацијата укажуваат на тоа дека процесот на развој на научните апликации кои го применуваат научниците е најмногу сличен на агилниот пристап за развој на софтвер. Во таа насока, оваа тема може да биде значајна за подобрување на процесот на развој на научните апликации. Резултатите од ова истражување се објавени во [120].

Главната причина за прифаќање на методите за агилен развој е да се создаде софтвер за пократко време и со намалени трошоци. Но, главното прашање кое се поставува при агилниот софтверски развој е дали се спроведува правилно тестирање.

Агилниот софтверски развој и тестирање може лесно да се вклопат во мал проект. Процесот на тестирање во случај на агилен развој на софтвер не е

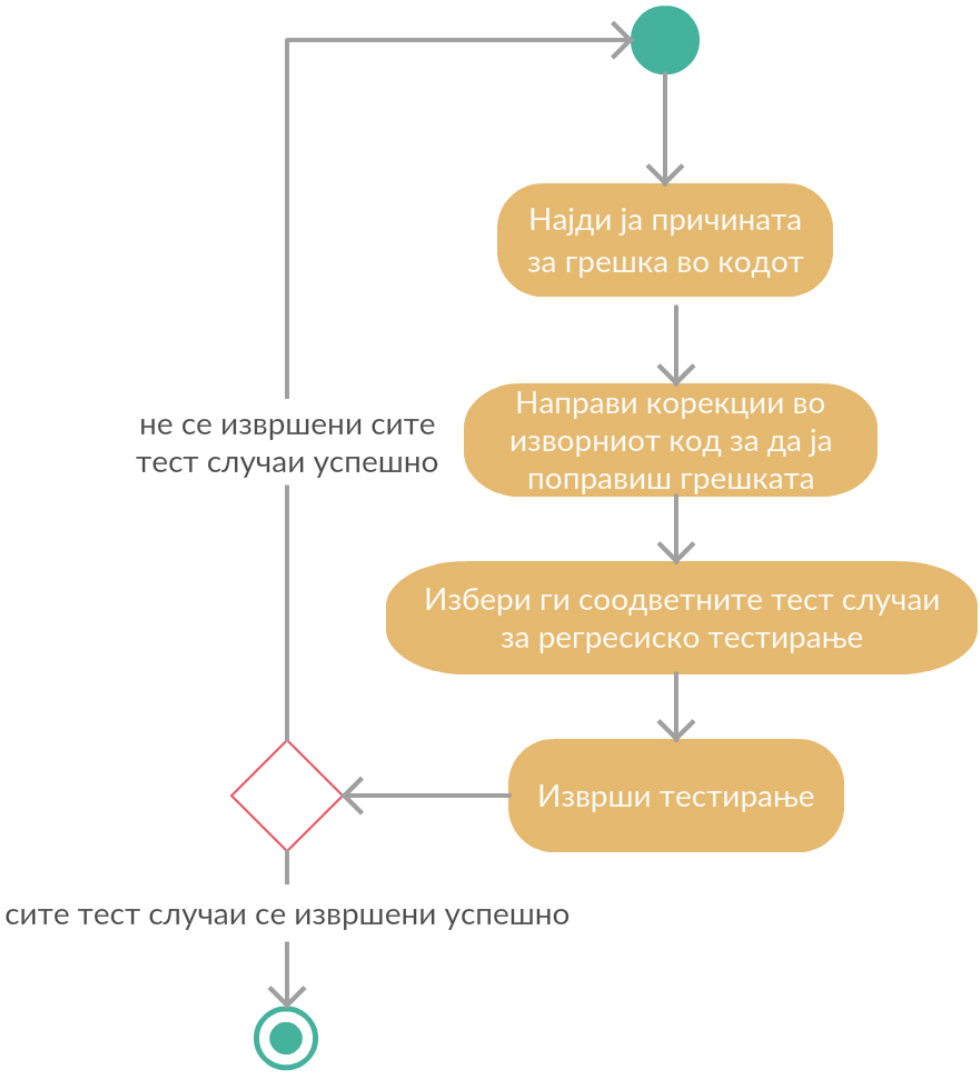


Слика 3.12 – Дијаграм на активности на регресиско тестирање кога се додава нова функционалност или се прави промена на постоечка.

унифициран и мора да се прилагоди на барањата на проектот, политиките на организацијата и способностите на тимот, со цел да се задоволат барањата.

Агилниот софтверски развој дозволува промени во барањата што значи дека може да се прават промени и во тест случаите. Затоа, потребна е подобра комуникација помеѓу развивачите, тестерите и корисниците со цел да се надминат проблемите и да се понудат пофлексибилни и оптимизирани решенија. Индивидуалците и нивните интеракции се најважниот дел од проектот, т.е. соработката помеѓу членовите на тимот е клучна за подобра средина за учење каде што новите членови можат многу да научат од постарите и поискусни членови на тимот [152].

Бидејќи за агилен развој се потребни мултифункционални тимови кои ги следат принципите на итеративни и инкрементални практики за развој, процесот на тестирање треба да биде ефикасен. Потребно е да се започне со рано



Слика 3.13 – Дијаграм на активности на регресиско тестирање кога ќе се пронајде регресиско тестирање.

тестирање и да се применува често тестирање. Мора да постои јасна дефиниција за тоа какви резултати се очекуваат од секое тестирање на крајот на секоја итерација (sprint). Тоа значи дека тестовите треба да се извршат пред имплементирањето на функционалностите на проектот на секоја итерација (sprint).

Клучниот фактор за успешно тестирање е соработката помеѓу корисниците, развивачите и тестерите. Тестерите треба да се дел од развојниот тим и активностите мора да се колку што е можно повеќе синхронизирани. На пример, додека тестерите работат на тест случаи, развивачите кодираат според барањата на корисниците. Бидејќи парадигмата за агилен развој гласи дека сите процеси мора да се завршат брзо, нема многу време за тестирање, што значи подготовката на податоци за тестирање треба да се заврши во фазата на планирање. Подобар пристап е овие податоци за тестирање да се подготват заедно со сите засегнати страни со цел да се задоволат функционалните барања и барањата за перформанси [152].

Бидејќи итеративниот развој е фокусиран на обезбедување на функционален софтвер на крајот од секоја итерација, мора да постои и нов софтверски дел со побаруваната функционалност. Времето потребно за секоја од итерациите зависи од големината на проектот и користените методи за агилен развој, но вообичаено потребни се од две до осум недели. Агилното софтверско тестирање им овозможува на тестерите да ја изразат својата креативност и да размислуваат во различни насоки [81].

Процесот на агилно тестирање не е унифициран и потребно е дополнително искуство и успешни анализи. Секој тим треба да ги дефинира своите стратегии и планови за софтверското тестирање [166].

Најтешкиот дел е да се создаде метод за тестирање со кој ќе се потврди дека решението на проблемот во тековната итерација е коректно. Тоа е уметноста на софтверското тестирање каде што не мора да бидат вклучени само формални методи и теории, туку и нови идеи и стратегии кои го олеснуваат процесот на тестирање.

3.6.1 Тестирање на софтвер при агилен развој наспроти традиционален развој на софтвер

Кога некоја компанија или истражувачка група го користи традиционалниот водопаден модел за развој на софтвер, тоа значи дека тестирањето се врши на крајот на процесот на развој. Тоа е вистински проблем кога зборуваме за квалитетен софтвер. Тестирањето на крај на развојот значи поголема можност за грешки во раните фази на развој на софтверот. Со тоа, се зголемуваат трошоците и се намалува останатото време наменето за развој.

Агилниот развој на софтвер може да помогне во намалување на грешките во раните фази од развој на софтверот. Позитивна страна е тоа што методите на агилен развој на софтвер може да се вклучат и во водопадниот модел. На пример, ако некоја компанија почнала да развива софтвер на традиционален начин со користење на водопадниот модел, методите на агилен развој може лесно да се вклопат во процесот [59].

Методите за агилно софтверско тестирање дозволуваат тестирање во текот на сите фази и тестирањето за извршува на мали единици код. Со тоа се обезбедува оптимизирано тестирање и задоволни корисници. Агилниот пристап е одличен пристап во многу ситуации, но не значи дека треба да се користи секаде. Тимот за развој заедно со тимот за тестирање треба да одлучат за методот кои ќе го користат во процесот на развој. Треба да најдат најсоодветен модел кои ќе обезбеди успешно развиен софтверски продукт.

3.6.2 Агилен развој кај обемни софтверски проекти

Методите за агилен развој на софтвер стануваат важен дел од еволутивниот развој на софтверскиот процес.

Најчестите принципи на агилниот развој на софтвер се следниве [152]:

- Агилниот развој на софтвер овозможува тестирање во раните фази од развој на софтверот и понатамошно тестирање паралелно со развојот на софтверот.

- Тестерите при агилниот развој на софтвер мора да се адаптираат на брзи промени.
- Тестирање мора да се извршува во секоја итерација.
- На крајот на секоја итерација, мора да се испорача функционална верзија на софтверот.
- Процесот на успешно тестирање бара блиска интеракција помеѓу корисникот и развивачот на софтверот со цел да се заштеди време преку избегнување на непотребни повторни дизајнирања и повторни имплементации.
- Прогресот во работата може да се пресмета со помош на планираните активности во зададен временски период.
- Континуирана интеграција е еден од најважните чекори во агилниот развој бидејќи тимот треба да понуди не само функционалност на индивидуалните задачи, туку и на бизнис логиката.
- Тестирањето треба биде автоматизирано бидејќи ќе овозможи побрзи резултати и поефикасна работа.

Користењето на овие принципи не гарантира успех. Многу работи зависат од структурата на тимот, големината на проектот и барањата на корисниците. Главниот фактор за успех е правилната комбинација на овие принципи кои ќе бидат подредени правилно со политиките за развој на компанијата.

3.6.3 Најдобри практики во методологиите на агилен развој на софтвер

Не постои унифициран процес за селекција на потребните практики кога станува збор за најдобрите практики, туку секој тим треба да ги избере вистинските практики за прогрес на проектот.

Првата работа која се поврзува со професионалното тестирање се тестерите. При агилното тестирање, сите пишуваат тестови, а професионалните тестери потребно е да напишат што е можно попрецизни и иновативни тестови, не само да смислат тест случаи. Професионалните тестери се вклучуваат со цел да се подобри координацијата помеѓу тестерите и развивачите на софтверот [198].

Второ, тестовите треба да се автоматизирани. Тоа значи дека тестовите ќе бидат завршени побрзо и поефикасно. За автоматизација потребна е селекција на вистинските алатки или изработка на сопствена алатка кои ќе ги задоволи потребите за тестирањето на проектот, ќе заштеди време и пари. Доколку алатката која одговара на потребите може да се најде на пазарот, помудро е да се искористи таа алатка отколку да се прави нова. Тестерите треба да знаат дека не треба сите тестови да биде автоматизирани. Предизвикот е да се најде баланс помеѓу мануелното и автоматизираното тестирање. За агилниот развој потребна е поголема автоматизација на процесот, но корисно е кога има некоја задача која се повторува или задача со регресиско тестирање. Нема потреба

од автоматизирање на тест случај која би било користена само еднаш бидејќи тоа време може да се искористи за некои нови тестови [18].

Наредно е планирањето на активности. Тоа значи планирање на времето потребно за развој, регресиското тестирање и исто така времето потребно за корекција на грешките. При агилен развој времето потребно за тестирање и кодирање треба да е еднакво. Доколку тимот знае дека некоја функционалност е потешка од некоја друга, тогаш предвиденото време треба да се пролонгира. Од друга страна, регресиското тестирање е алоцирано како глобален временски период бидејќи овие тестови се извршуваат на крајот на секоја итерација. Разликата во времето помеѓу вкупното време потрошено на регресиското тестирање и итерациите е мала и тоа е причината зошто времето за регресиско тестирање се смета како глобално. Времето потребно за поправање на грешките исто така се смета за глобално. Тоа е така бидејќи не може да се предвиди колку време би било потребно за пронаоѓање и поправка на некој дефект. Планирањето на грешките не е добра идеја бидеќи не можеме да предвидиме некоја грешка и колку време би ни било потребно за поправка. Главната парадигма во агилниот развој во врска со поправка на грешките е тие да се коригираат колку е можно побрзо [198].

3.6.4 Автоматизација на тестирањето при агилниот развој на софтвер

Автоматизацијата на методите за агилно софтверско тестирање е примарно наменета да ја минимизира мануелната работа вклучена во извршувањето на тестирањето и да се добие поголема покриеност со поголем број на тест случаи. Автоматизираните тестови во агилното тестирање се вообичаено тестирања на софтверски делови кои се одговорни за квалитетот на најмалите можни модули [106].

Пишувањето на тестовите е скоро една третина од тестирањето. Бидејќи технологијата за автоматизација не напредува според очекувањата, алатките за автоматско генерирање на тестови понекогаш генерираат преобемни множества за тестирање што ги намалува придобивките од автоматизацијата. Алатките треба да се лесни за користење и мора да овозможат едноставен кориснички интерфејс. Сите скрипти мора да се подготват добро за да се добие точниот резултат. Од големо значење е да се поделат долгите тест случаи во помали бидејќи помалите тест случаи полесно се одржуваат. Развивачите треба исто така да помогнат во процесот на автоматизација преку давање на дополнително мислење за решавање на проблемите. Едно од клучните прашања е како да се избере вистинската алатка за автоматизација или како да се направи своја алатка која ќе ги исполнува потребите на проектот. Постојат голем број на алатки достапни на пазарот, но знаењето и искуството на тимот треба да помогне при изборот на најсоодветна алатка. Одлуката зависи од големината на проектот, времето и трошоците [106].

Следниве чекори можат да помогнат при тестирање при агилен развој на софтвер со цел да се постигне оптимизација на процесот на тестирање:

- Се дефинираат функционални барања.

- Тие се разгледуваат тимот за гаранција на квалитетот.
- Во фазите на дизајн и имплементација, корисничките приказни се пишуваат и се разгледуваат од корисникот.
- Спецификациите за барањата се обновуваат со нови промени.
- Кога започнува имплементацијата, тест случаите и стратегиите се подготвени сите треба да се докуметирани и разгледани од тимот на корисници.
- Во текот на имплементацијата, тимот за тестирање одлучува дали кодот за тестирање може да се примени врз софтверот.
- Процесот итерира додека софтверот не е целосно прифатен.

Никогаш не можеме да бидеме сигурни дека софтверот е 100% без грешки кога се предава на клиентот. Агилниот метод е најдобар пристап за софтвер чии барања се менуваат брзо. За различни проекти потребно е да се дефинираат соодветни стратегии да се обезбеди квалитет на системот. Понекогаш, подобро решение е хевристички пристап кој се базира на претходно искуство за тестирање.

Во областа на научните апликации спаѓаат и софтвери за кои задолжително мора да се обезбеди документација од тестирањето за да се одобри користењето на софтверот. Примери за такви софтвери се од областа на медицината, фармацевската индустрија и сл. Оттаму, примената на сите препораки при агилен развој можат значително да го подобрат развојот на овие софтвери.

3.7 Управување со промени и контрола на верзии на софтверот

3.7.1 Идентификација на можните проблеми

Со оглед на фактот што научниците обично не ги применуваат практитките на софтверското инженерство во процесот на развој на научните апликации [182, 122], тие не сметаат дека управувањето со промените на софтверот за контрола на верзиите на софтверот се важни фактори за развој на квалитетни апликации. Веќе прифатените практики од софтверското инженерство кои најчесто се користат во процесот на развој на комерцијални апликации, не се прифатени во развојот на научниот софтвер. Овие практики се првенствено насочени кон употреба на одредени стандарди, шаблони, методи, итн. Истражувањето покажа дека научниците не се запознаени со управување со промените и процесот на контрола на верзии и соодветните алатки.

Промените при развој на научните апликации треба да се направат во согласност со тековните практики на научен развој на софтвер, но потребно е да се вклучат практиките на софтверското инженерство. Овој пристап е важен бидејќи научниците може полесно да ги прифатат промените и да се воспостави поуспешна соработка меѓу софтверските инженери и научниците.

Процесот на контрола на верзиите на софтверот треба да се направи со користење на софтверски алатки кои ќе обезбедат корисничко пријателска околина и ефикасност. Една од најважните работи кои треба да се земат в предвид при изборот на алатката за контрола на верзиите е способноста да се обезбеди координација помеѓу промените направени на исти делови од кодот делови во исто време и визуелна претстава за промените направени во секоја верзија во однос на претходната.

Непримену̀вањето на практики на софтверското инженерство во сите фази на развој, особено при управување со промените кое се јавува како резултат на корекција на грешки во кодот или надградба на софтверот води до создавање на ниско квалитетни апликации и нерешени проблеми. Бидејќи не постојат вистински корисници, тестирањето на прифаќање треба да се замени со потврдување преку резултатите добиени од експерименти. Исто така, координацијата помеѓу научниците и софтверски инженери е важен фактор за успешно управување со промените.

3.7.2 Модел за управување со промени

За успешно прилагодување на практиките за управување со промени на софтверот и контрола на верзиите на софтверот треба се утврдат карактеристиките на научните апликации и да се идентификуваат разликите во процесот на развој помеѓу комерцијалните и научни апликации. Следно, постоечките практики на управување со промените и контрола на верзиите на софтверот треба да се изменат со цел да се покријат сите специфики на научниот софтвер. Покрај тоа, треба да се следат и основните концепти на развој, кои се веќе прифатени како стандарди за квалитет. Измените мора да овозможат флексибилен процес на управување со промените и контрола на верзиите на софтверот. Софтверот за контрола на верзиите на софтверот треба да биде изменет во согласност со потребите на апликацијата. Резултатите од ова истражување се објавени во [119].

На слика 3.14 е претставена модификација на моделот за управување со промени претставен во [73]. Овој модел може лесно да се адаптира за управување со промени при развој на научен софтвер.

Направени се следниве измени:

- чекорот во втората спирала „дефинирање на група од интерест”, во делот „Евалуација на алтернативни решенија, идентификација и справување со ризици” треба да се отстрани (не постои посебна група од интерес кај научниот софтвер);

- чекорот во четвртата спирала „тестирање за прифаќање на решението” во делот „Развој” треба да се замени со „Споредба на резултатите од софтверот со резултатите добиени од физички експеримент” или доколку тоа не е можно со оценување на резултатите со користење теорија или универзално прифатените решенија (не постојат вистински корисници за тестирање на прифаќање на решението);

Слика 3.14 – Модел за управување со промени на софтвер

3.7.3 План за управување со промени кај научните апликации

Планот за управување со промени е важен чекор кој треба да се направи пред да започне процесот на кодирање. При развој на научен софтвер создавањето на ваков план е неопходно бидејќи промените се составен дел од развојот и тие обично се јавуваат во неочекувано време.

Една можна структура на документот во кој се опишува планот за управување со промени би била следнава:

- Опис на проектот. Ова е делот каде се објаснува проблемот што треба да се реши, а исто така се опишуваат и некои детали за неговото решение од логички и технички гледишта.

- Речник. Во овој дел треба да се дефинираат термините кои се користат во целиот документ. Зборовите кои се користат во научниот домен треба да бидат вклучени исто така. Речникот е особено корисен за софтверски инженери кои може да се приклучат подоцна во тимот за развој.

- Улоги при справување со промените. Во овој дел треба да се наведат одговорностите на членовите на тимот за развој кои се дел од процесот на управување со промени. Покрај тоа, секој член мора добие конкретни задачи за различни видови на промени, како што се: идентификација на промени, спроведувањето на промените, ризик за идентификација, итн.

- Пристап за управување со промени. Во овој дел се опишува процесот на справување со промените, односно приодот кој треба да се користи

за справување со било која промена. Промените треба да бидат поделени според нивниот вид. Ако има повеќе различни видови, треба да се предложи одредено решение за секој тип на промени. Процесот на справување со промени треба да бидат опишан во детали, почнувајќи од барање за промена до нивната реализација. Овој дел мора да биде надополнуван постојано, при одобрување и имплементирање на промена. Оваа информација може да биде корисна кога нов член се приклучува на тимот за развој.

- Опис на промена. Во овој дел се опишува во детали проблемот со кој треба да се реши со конкретна промена.

- Влијание на промена. Влијанието на промената мора да се наведе со цел да се воочат последиците кои може да се појават како резултат на промена на код.

- Статус на софтверот пред и по спроведувањето на промената. Овој дел треба да се користи за да се покажат разликите помеѓу сегментите на кодот кои се опфатени со актуелната промена. Доколку се користи систем за контрола на верзиите на софтверот, тогаш тоа може да помогне во процесот на прикажување на статусот на кодот пред и по промената.

- Имплементација на промената. Тука се објаснува техничкото решение на имплементација на промената. Овој чекор е особено корисен за софтверските инженери и научници, кои ќе се приклучат на процесот на развој подоцна.

- Идентификација на ризици. Овој дел е наменет за да се идентификуваат сите ризици кои може да се појават како резултат на спроведувањето на промени. Секој од можните ризици треба да се опише во детали. Потребно е да бидат напишани следниве информации: тип на ризик, погодени сегменти од кодот, модули или имплементациска логика и идните промени кои може да се случат.

- Валидација на промена. Валидацијата на софтвер е процесот на тестирање. Овој дел треба да ги опише методите на тестирање кои се користат за проверка на функционалноста на системот по промената која се спроведува, а исто така треба да бидат претставени и конечните резултати. Ако се спроведува физички експеримент, тогаш треба да се даде заклучокот на споредба помеѓу експериментални резултати и резултатите добиени од апликацијата.

  Добрата стратегија за справување со промените овозможува повеќе контрола на активностите за развој на апликацијата. Покрај тоа, програмерите учат да стекнуваат основни навики за примена на практиките на софтверското инженерство кои се дел од стандардите за квалитет на софтверот.

3.7.4 Избор на алатки за контрола на верзии на софтверот

Не постои универзално прифатена алатка за контрола на верзиите на софтверот при развој на научни апликации. За да се направи добар избор, развојниот тим мора да ги дефинира потребите и целите на научната апликација. Важно да се наведат и сите ограничувања, бидејќи тоа помага во процесот на селекција. Искуството на членовите од научната заедницата која се занимава со истражувања од сличен тип може да помогне при изборот на алатката. Изборот на алатка за контрола на верзиите на софтверот зависи од многу фактори како што се: големината на апликацијата, бројот на членовите во тимот, потребата за повеќе детали, типот на извештаи, визуелно претставување на промените, итн. Следниве алатки за контрола на верзиите на софтвер се со отворен код и имаат поддршка за различни оперативни системи:

- Git - дистрибуирана алатка за контрола на верзиите на софтвер достапна за повеќе платформи: Linux, Windows, Mac OS X, Solaris. Поддржува евтини локални разгранувања, прилагодлив интерфејс, и повеќе работни процеси [8].
- Subvesrion - централен систем за контрола на верзиите кој е дел од фондацијата Apache софтвер и обезбедува сигурна, едноставна и безбедна средина [9].
- CVS - клиент-сервер систем за контрола на верзии за Unix и Windows оперативни системи. Нуди следење на изворен код и документи, како и флексибилно запишување на промените во база на податоци [7].

CVS е постар систем и ретко се користи и денес. За корисниците кои сакаат побрзо и дистрибуирано решение се препорачува Git. За корисниците кои преферираат централизиран пристап и организација се препорачува Subversion.

Глава 4

Рамка за развој на научни апликации

Во ова поглавје се дефинира и евалуира рамката за развој на научни апликации. Рамката опфаќа модел за развој на научни апликации и модел за квалитет за научни апликации. Моделот за квалитет вклучува и квантитативна евалуација на атрибути кои се вреднуваат во секоја од фазите на развој на научните апликации.

4.1 Дефиниција на рамка за развој на научни апликации

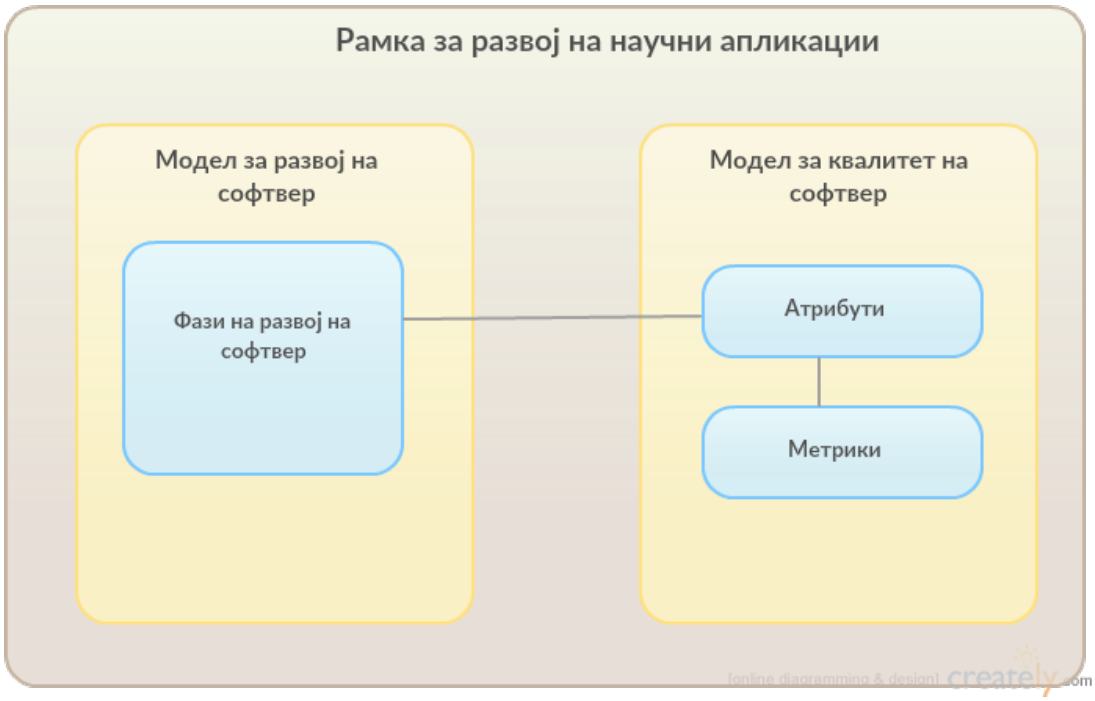
Според енциклопедијата „Британика" поимот рамка е дефиниран како основна концептуална структура (од идеи) или скелет, структурна рамка [37]. Според [99], рамката обезбедува сеопфатно множество на елементи (конструкции) и правила за нивна примена, кои служат како основа за создавање на апликации.

Предложената рамка за развој на научни апликации, прикажана на слика 4.1 опфаќа два важни аспекти за развојниот циклус на апликациите: моделот за развој и моделот за квалитет на апликациите.

Модел за развој на софтвер претставува карактеризација на процесот на развој на софтверот. Моделот се користи како водич или рамка за организација и структурирање на активностите за развој на софтверот. Моделите за развој на софтвер, исто така, се користат за интегрирање на задачи од развојот и техники за користење на дадено множество на алатки од софтверското инженерство или околини за развој [179].

Квалитетот на софтвер се дефинира како степен до кој софтверот поседува посакувана комбинација од атрибути [50]. Атрибутите придонесуваат за квалитет на софтверот. Во секоја фаза треба да одреди множеството на атрибути кое е значајно за таа фаза. Метриките се поврзани со атрибутите и претставуваат функции кои на влез примаат податоци од софтверот, а на излез даваат нумеричка вредност. Метриката може да се интерпретира како степен до кој софтверот поседува атрибут кој влијае на неговиот квалитет [4].

Моделот за квалитет за софтвер опфаќа множество на атрибути кои се евалуираат со метрики. За да се евалуира квалитетот на научниот софтверот, во секоја фаза од развојот на научната апликација се вклучува множество на



Слика 4.1 – Глобален поглед на рамката за развој на научни апликации

атрибути кои имаат влијание на квалитетот на таа фаза. Исти атрибути може да влијаат на квалитетот во различни фази од развојот на софтверот.

4.2 Модел за развој на научни апликации

4.2.1 Карактеристики на моделите за развој на софтвер

Анализата на различните модели на развој на софтвер помага да се воочаат разликите и да се избере соодветниот модел или комбинацијата од модели за развој на научните апликации.

Постојат неколку процесни модели за развој на софтвер презентирани во стандардната литература кои претставуваат апстрактна репрезентација на процесот на развој. Како најпознати и најкористени модели на развој на софтвер може да се издвојат следниве: водопаден модел, итеративен модел, V-модел, спирален модел, инкрементален модел, модел на прототипи, агилен развој на софтвер, итн. Исто така, постојат и некои варијанти и комбинации на овие модели [179, 151].

Водопадниот модел на развој на софтвер е еден од најстарите традиционални начини на развој на софтвер. Тој е составен од неколку развојни фази кои не се преклопуваат: Дефинирање на барања, системски и софтверски дизајн, имплементација (кодирање), тестирање, функционирање и одржување. Овој модел лесно се имплементира и разбира, но тој не ја отсликува реалноста. Водопадниот модел забранува враќање во претходната фаза на развој, но оваа

операција е многу скапа. На пример, промена во барањата значи промена во сите последователни фази.

Итеративниот модел на развој на софтвер се карактеризира со побрзи развојни фази, а тоа обезбедува поголема флексибилност. Една фаза од развојот се состои од чекорите за развој кај водопадниот модел, но само мал дел од софтверот се развива. Испораките од секоја фаза нудат можност на корисниците да дадат повратна информација на крајот на секоја итерација. Тоа им помага на развивачите на софтверот да ги коригираат грешките во почетокот од процесот на развој.

V-моделот обезбедува секвенцијален проток на активности исто како водопадниот модел, но акцентот е ставен на тестирањето, односно планот за тестирање се прави пред фазата на имплементација. Планот за интеграциско тестирање е создадена во фазата на дизајн на високо ниво. Тестирањето започнува кога кодирањето е завршена. Овој модел е погоден за мали проекти, но исто и како водопадниот модел, не се испорачува софтвер пред фазата на имплементација.

Спиралниот модел на развој на софтвер всушност претставува итеративен развој, но посебно внимание се посветува на анализа на ризиците. Секоја итерација резултира во прототип, а се состои од четири фази: планирање, анализа на ризик, инженерство и евалуација. Овој модел е погоден за големи проекти, но анализите на ризици може да бидат скапи.

Инкременталниот модел на развој на софтвер е комбинација на водопадниот модел на развој на софтвер итеративниот модел. Системот се развива во инкременти и може да биде дистрибуиран периодично до корисничките заедници. Системот се развива онолку колку што е потребно во даден момент, а подоцна се додава на врвот на сè она што постои до тој момент. Овој пристап е проширен од страна на софтверските прототипови.

Методот на прототипови обезбедува функционалност на системот во раните фази на развој. Повеќе напор се вложува во фазите на спецификација и анализа на барања. Постои директна комуникација помеѓу корисниците и програмерите и учество на корисниците во дефинирањето на функционалноста на системот и негова евалуација.

Агилниот развој на софтвер вклучува множество на методи и практики кои се базираат на вредности и принципи дефинирани во Манифестот за агилен развој [70]. Целта на агилниот развој на софтвер е да се даде предност на индивидуите и интеракциите отколку на процесите и алатките. Позначајно е да се добие функционален софтвер отколку да се добие преобемна документација. Соработката со клиентите е секогаш пред склучувањето на формални договори. Следењето на планот се занемарува во случај кога се појавува промена во софтверот.

4.2.2 Нов модел за развој на научни апликации

Земајќи ги в предвид карактеристиките на научните апликации и карактеристиките на постоечките модели на развој на софтвер во оваа секција се предлага нов модел за развој на научен софтвер. Понудените модели за развој се дизајнирани за развој на комерцијални апликации и за да се приспособат

за научни апликации, треба да се направат измени во некои фази во развојот. Причините за промени се базираат на неколку клучни разлики помеѓу комерцијалните и научните апликации:

- Резултатите од научни апликации не може секогаш да бидат споредени со теоретски заклучоци, туку тие мора да бидат потврдени со споредба на резултатите добиени од реални физички експерименти;
- Научните апликации се дизајнирани за специфична научна заедница, а не за комерцијална употреба. Тоа значи дека не постојат корисници кои што може да ги специфицираат барањата на софтверот и да ја оценат функционалноста на апликациите;
- Различните сфаќања и практики на развој на научниците и софтверските инженери;
- Грешките кај научните апликации може да се јават исто така и при дефинирање на математичките модели [107].

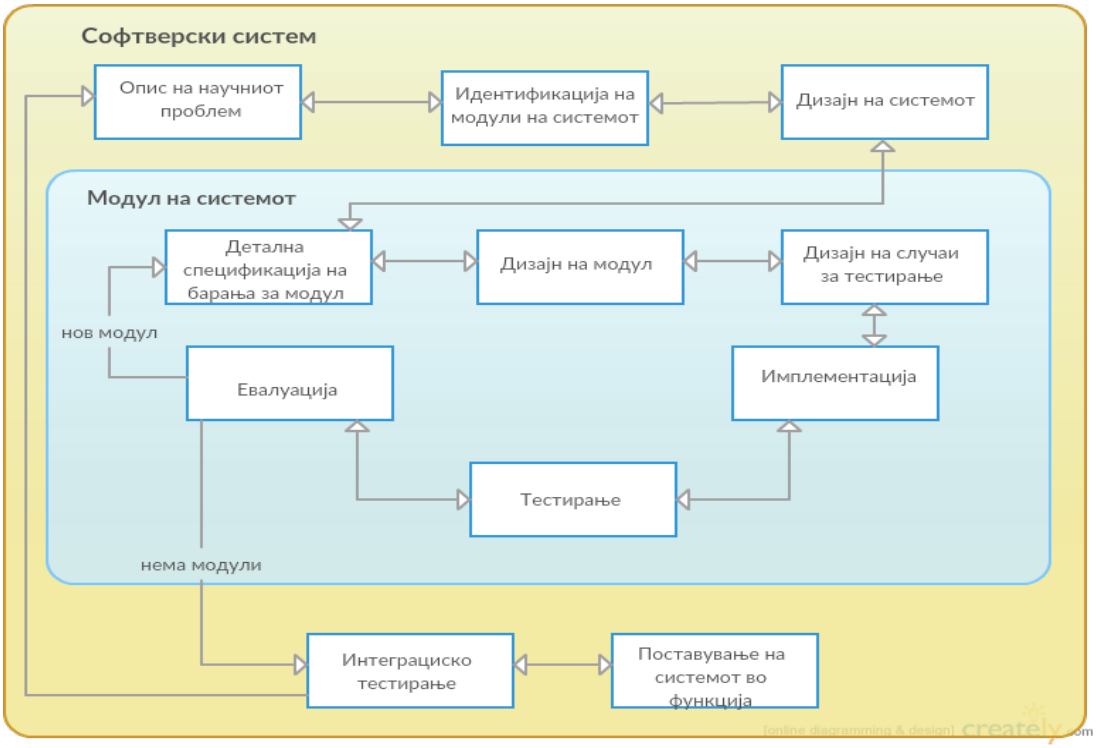
Целта на предложениот модел за развој на научни апликации е да се променат сегашните практики за развој на научниците со тоа што ќе се посвети повеќе внимание на оние фази на развој кои воопшто не се практикуваат од страна на научниците, како и да се подобри квалитетот на процесот на развој на софтверот.

Моделот за развој на научни апликации го зема в предвид фактот дека научните апликации се комплексни и обемни апликации и според тоа се заклучува дека најдобро би било да се развиваат постепено со идентификација на помали целини - модули (понекогаш модулите се дистрибуираат за да се извршат на различни сервери) и многу е важно секој модул да функционира што е можно повеќе независно, без оглед на другите модули. Модулите често се искористуваат и во други апликации. Функционалноста на секој модул треба да се тестира веднаш по неговата имплементација бидејќи тој модул може да биде вклучен во друг модул кој ќе се развие подоцна.

Грешките кај научните апликации може да се јават и во фаза на дизајн на математичкиот модел и во фаза на имлементација на кодот. Поради тоа, моделот за развој мора да обезбеди враќање во претходните фази на развој. Предложениот модел вклучува особини од повеќе процесни модели (итеративен, инкрементален, агилен, модел на прототипи), промена во постоечките фази (различна евалуација) и дополнително при развој на секој модул додава нови чекори: дизајн на модул, дизајнирање на случаи за тестирање пред имплементација на модулот. Спецификацијата на системот која е предложена во овој модел е базирана на практиките за софтверска спецификација предложена од IEEE [49], но сепак е прилагодена за развој на научни апликации. Тоа значи дека некои делови се занемарени (на пр. деловите кои се однесуваат на корисници), а дополнително се додадени нови делови.

Моделот за развој на научни апликации е прикажан на слика 4.2. Развојот на научните апликации започнува со фазата на опис на научниот проблем. Од секоја фаза може да се премине во претходната, освен од фазата на интеграциско тестирање, во која доколку се пронајде грешка треба да се вратиме

во фазата на глобална спецификација на барања. Делот за развој на модул може да се повторува повеќе пати во зависност од бројот на модули кој ќе се идентификува во апликацијата. Детали за секоја од фазите се дадени во следните поглавја од оваа секција. Резултатите од ова истражување се објавени во [124].



Слика 4.2 – Модел за развој на научни апликации

4.2.3 Фази на развој на научна апликација

Во ова поглавје се дава детален опис на сите фази од моделот за развој на софтвер прикажан на слика 4.2.

Опис на научниот проблем

Фазата на опис на проблемот опфаќа дефинирање на научниот проблем кој подоцна треба да биде преведен во алгоритам во одреден програмски јазик. Дефинирањето на научниот проблем е потребно за споредба на резултатите, како и дообјансување на алгоритмот. Во овој дел треба да се објасни накратко теоријата на проблемот кој се решава и да се дефинира математички.

Резултатот од оваа фаза служи како потсетник за тоа што е главната цел на апликацијата. Документот треба да содржи: цел на апликацијата и краток опис за тоа што се очекува како краен продукт, како и дефиниција на основни поими и кратенки кои ќе се користат во текстот.

Идентификација на модули на системот

Идентификацијата на модули на системот претставува многу значајна фаза од развојот на научната апликација. Во оваа фаза треба да се идентификуваат независните функции на системот и да се дефинираат како модули. Во зависност од поделбата по модули, понатаму зависи успешноста на развојот на апликацијата. Независните модули дефинирани во оваа фаза ќе може да се тестираат и евалуираат самостојно. Доколку сите модули не може да се утврдат на почеток, постои можност да се дополни оваа листа. Во оваа фаза треба да се направи и организација на активностите за развој на модулите.

Дизајн на системот

Фазата за дизајн на системот нуди преглед на барањата за хардвер и софтвер, архитектура на системот, модули, компоненти, податочни модели, интерфејси и податоци за задоволување на барањата утврдени во претходните две фази.

Архитектурата на системот ја опишува структурата и однесувањето на системот. Логичката архитектура преставува апстрактна репрезентација на системот и е создадена од моделирање на системот. Моделирањето на системот може да се направи со користење на алатки кои обезбедуваат услови за графичко моделирање, каде се специфицираат текови на податоци, влезови и излези на системот (пример за вакво моделирање е прикажан во секцијата 3.2.4). Ваквите системи за моделирање на работни текови се користат за проектирање, автоматизација, контрола и управување со сложени протоци на податоци и процеси. Научните заедници може да ги користат овие системи за спроведување на експерименти со големо множество на податоци и пресметки кои се извршуваат на далечина, додека бизнис компаниите може да ги користат за да го автоматизираат процесот на производство и да се избегнат непотребните задачи. Влезните и излезните процеси на системот, а исто така и верификацијата и обработка на податоците се дел од физичкиот дизајн на системот.

Сумарно, дизајнот на системот е фаза од развојот, каде се анализира дефинира и имплентира архитектурата на системот. Во овој дел, се дефинираат хардверот (машини, меморија и сл.) и софтверот кој ќе се користи при развој на системот (оперативен систем, алатки и сл.). На пример, потребите на апликацијата може да вклучуваат повеќе јадрени процесори, грид, компјутери со високи перформанси или некои библиотеки кои обезбедуваат паралелно извршување, компајлери, интегрирана развојна околина, платформи и оперативни системи. Дополнително, добро е да се специфицира верзијата на потребниот софтвер за да може подоцна да се потврди функционалноста на системот за таа верзија.

Исто така, доколку има ограничувања на системот како целина треба да се додадат во документот.

Спецификација на барања за модул

Природата на научниот софтвер наметнува пишување на спецификација на барања во модули. Фазата на спецификација на барањата е прва фаза од развојот на еден модул. Финалниот производ што произлегува од оваа фаза треба да биде формално напишан документ кој содржи листа на дефинирани функционални барања за модулот. Потребните полиња кои треба да се внесат за секое барање во спецификацијата на барања се дадени во продолжение:

- ИД барање - единствен идентификатор на барањето;
- Верзија - тековната верзија на барањето (доколку се извршен било какви корекции);
- Име - кратко име кое го опишува барањето;
- Опис - опис на функционалноста
- Историја на промени во секоја верзија на барањето.

Фазата на спецификации на барањата е од клучно значење за отстранување на можните грешки во следните фази. Тоа исто така помага во дизајнот на тест случаи подоцна. Анализата на барања спречува одложена реализација на задачите, како и ненавремено и нецелосно тестирање.

Доколку сите барања за еден модул не се познати во почетокот на процесот на развој на модулот, моделот за развој овозможува да се додадат подоцна. Спецификацијата на барања за сите модули може да бидат напишани во еден документ кој постојано ќе се надополнува и ќе биде подложен на промени.

Дизајн на модул

Во оваа фаза, доколку природата на системот го наложува тоа, може да се вклучи формална спецификација на модулите како што е примерот во поглавјето 3.1.4. Особено значајна работа во овој дел е да се дефинира соодветен програмскиот модел или решение, доколку постои. Идентификацијата на веќе прифатено оптимално решение или програмски модел за решение на проблемот го намалува потребното време за развој и ја подобрува ефикасноста на алгоритмот. Пример за користење на програмскиот модел Map-Reduce е даден во поглавјето 3.3.3.

Дизајн на тест случаи за модул

Дизајнот на тест случаи е тесно поврзан со спецификацијата на функционалните барања. Добро специфицираните барања доведуваат до разбирливи и добро специфицирани тест случаи со што се обезбедува поголема прегледност на карактеристиките кои треба да бидат тестирани. Спецификацијата на тест случаи во развојот на научен софтвер е тешка задача, бидејќи понекогаш очекуваниот резултат не е познат во оваа фаза. Во тој случај, добра практика е да се анализира колекција на слични проблеми.

При развојот на научни апликации, потребните параметри за дефинирање на тест случаите или се познати или може да се пресметуваат, на пример, со аналитички пристап. Тест случаите треба да бидат специфицирани со користење на стандардизирани форми, специфични техники базирани на модели, анализа на граничните вредности, анализа на изворниот код, итн. Онаму каде што не е можно да се определи точната вредност може да се наведе ранг на вредности. Потребните полиња за дефинирање на еден тест случај се дадени во продолжение:

- ИД - единствен идентификатор на тест случајот;
- Име - име на тест случајот;
- ИД на барањето за кое е наменет тест случајот;
- Предуслови - услови кои мора да бидат исполнети со цел да се изврши тест случајот (резултати од други тестови или некои дополнителни услови).

За секој чекор при извршувањето на тест случајот се запишува:

- Краток опис на акцијата
- Очекуван резултат;
- Статус на тест случајот (Успешен/Неуспешен);
- Коментар.

Креирањето на тест случаите може да се направи со користење на два различни пристапи: статички пристап кој генерира влезови од модел на системот (тестирање базирано на модели) и динамички пристап кој генерира тест случаи со секое извршување на програмата.

Техники за дизајн на тест случаи базирани на структура вклучуваат: техники на покриеност на кодот, техники на покриеност на гранки во кодот, услови, итн. Документите креирани за спецификација на код, вклучувајќи ги и резултатите од анализата на граничните вредности се опции за дизајнирање на тест случаи.

Во случај на редундантни тест случаи, потребно е да се искористи добра техника на селекција со која ќе се елиминираат непотребните тест случаи и ќе се даде приорирет на тест случаите со подобра можност за детекција на грешки.

Промените во барањата или извореният код предизвукуваат промени и во тест случаите. За полесна идентификација на тест случаи кои се поврзани со одредена промена во кодот, потребно е кодот да се организира во помали независни модули. Ако од еден тест случај зависат други тест случаи и промената е извршена во тој тест случај, потребно е повторно да се изминат сите тест случаи кои се зависни од тој тест случај.

Тест случаите треба да бидат специфицирани пред почетокот на фазата на пишување на код. Редундантните тест случаи треба да се отстранат со

правење паметен избор на тест случаи со што ќе се намали бројот на тестови. Можните конфликти со креираните тестови мора да се решат со промена на тестовите. Тест случаи со поголема можност за грешка треба да се означат со повисок приоритет. Селекцијата треба да се направи паметно за да може да се покријат задоволителен број на случаи. Во одредени случаи ова се прави рачно, но постојат и алатки за селекција на случаи за тестирање. Не постои универзален прифатен начин за селекција на случаи на тестирање. Искуството на научниците најчесто влијае во овој случај. Генерални препораки кои треба да се земат в предвид се: да се изберат случаи за тестирање со различно множество на податоци, со големо множество на податоци, оние кои се зависини од други тест случаи, случаи кои побаруваат специјални податоци.

Имплементација

Повторното искористување на веќе постоечки компоненти или алгоритми во процесот на развој на масивни системи е клучен фактор за намалување на времето потребно за програмирање. Исто така, постои можност да се развијат одредени веб сервиси или библиотеки со функции за решавање на некои веќе познати проблеми.

Паралелизацијата од кодот е исто така многу корисна практика во процесот на развој на научен софтвер и тоа доведува до оптимизација на кодот. Вообичаено, паралелизацијата се постигнува со користење на дополнителни библиотеки, како што е OpenMPI [67].

Следниве практики ја намалуваат можноста за грешки и ја подобруваат разбирливоста на кодот:

- Долги описни имиња за променливи, функции, објекти, итн
- Пишување коментари во кодот;
- Избегнување на декларација и дефиниција на променливи кои не се користат;
- Справување со исклучоци (делење со нула, недостаток на влезна датотека, недостаток на меморија);
- Соодветно вовлекување на линиите код.

Тестирање на модул

Најголемиот проблем при тестирањето на научните апликации е недостатокот од реални податоци за тестирање.

Процесот на тестирање започнува по создавањето и селекцијата на тест случаите и имплементација на кодот на една индивидуална целина од софтверот. Постојат многу различни методи кои се користат за проверка на коректноста на софтверот.

Еден од можните начини на тестирање на научниот софтвер е тестирање на софтверот како бела кутија со кое се врши проверка на изворниот код. Во предложениот модел се генерираат тестови за секој модул. Во овој дел,

најмногу внимание се посветува на тестирање на функционалните барања опишани во спецификацијата на барања.

Функционалното тестирање и тестирањето на изворниот код може да се врши со користење на алатки кои овозможуваат автоматизација на тестирањето и проверка на изворниот код. Постојат различни рамки за тестирање за создавање и извршување на тестови.

На пример, апликациите кои се развиени во програмскиот јазик C може да се тестираат со овие рамки: Cunit, Check, CuTest, итн. Секоја од овие рамки е добро документирана и лесна за употреба. Сите од нив обезбедуваат методи и структури, а корисникот е одговорен само за вметнување на проверки во кодот. Исто така, постојат алатки кои можат да вршат активности како што се: анализа на контрола и тек на податоци, статички анализи на код, проверка на стандарди за кодирање, итн.

Тестирањето на модули е завршено кога ќе се постигне покриеност на изворниот код преку тестови и кога тестовите на сите модули ќе поминат успешно. Потоа може да се премине кон интеграциското тестирање. Кога ќе се направи промена во некој дел од кодот, сите тестови засегнати од промената мора да се извршат повторно.

Евалуација

При евалуација на научни апликации, најчесто добиениете резултати се споредуваат со резултатите добиени од физички експерименти или се потпираат на некои теориски докази (математички модели). Доколку добиеме различни решенија, не може да се утврди дали грешката е во кодот или моделот. Притоа, некогаш е невозможно да се направат физички експерименти заради безбедност и цена.

Вообичаено, апликациите се евалуираат со помош на некои познати решенија кои ги опфаќаат поширок спектар на можни вредности и случаи. Ако проблемот за решавање е релативно нов, тогаш научниците ја следат логиката на резултатите и трендовите кои ги очекуваат. Резултатите се оценуваат со споредување на резултатите од апликации развиени од други научници кои се дел од истиот истражувачки домен. Поради немањето практика за спецификацијата на барања, тешко е за веќе развиените апликации да се најде токму онаа со која што може да се евалуираат резултатите. Доколку за проблемот може да се добијат аналитички решенија, резултатите може да се споредат со нив, но добивањето на аналитички решенија не е секогаш можно.

Доколку не може да се спореди со веќе постоечко решение постои метаморфно тестирање [41] со кое се проверува дали апликацијата се однесува според дефинирано множество на особини (метаморфни релации). Релациите специфицираат како промена на влезен параметар влијае на промена на резултатот.

Кај некои научни апликации може да се добијат само приближни решенија. Притоа треба да воведе толеранција за грешка бидејќи честопати нумеричките решенија се заокружуваат. Грешките не треба да се бараат само во кодот, туку и во математичкиот модел.

Искуството за развој и искуството од минатите грешки може да им помогне на научниците да најдат грешки и да го подобрат процесот на дизајн тест случаи.

Интеграциско тестирање

Разликата меѓу тестирањето на индивдуални делови од кодот (модули) и интеграциското тестирање е дека кога се врши тестирање на модул фокусот се става на анализа на имплементираните алгоритми и употребата на податоци, а кога се врши интеграциско тестирање, тестирањето се сведува на анализа на односите меѓу интерфејси и модули.

Главната цел на интеграциското тестирање е да се потврди дека постои правилна комуникација меѓу софтверските компоненти, што би значело да немаат неусогласеност со нивните интерфејси.

Интеграциското тестирање врши проверки на функционалноста на системот како целина, со тестирање на врските помеѓу модулите. Инетграциско тестирање треба да се направи кога сите тестови на модулите се завршени и резултатите се евалуирани. Тестирањето на системот бара имплементација на функционалното тестирање за сите барања и врски до одредени тест случаи.

Поставување на софтверот во функција

Поставувањето на софтверот во функција е активност која се врши на крајот на процесот на развој, по завршувањето на интеграциското тестирање. Овој процес содржи голем број на меѓусебно поврзани активности, како што се: испорака на софтверот на крајот на процесот на развој, конфигурација на софтверот, инсталација на околината за извршување и активација на софтверот. Поставувањето на софтверот во функција, исто така, вклучува и следење, деактивирање, ажурирање, повторна конфигурација, адаптација, повторно поставување и деинсталација на софтверот.

Поставувањето на софтверот во функција е сложен процес, особено кога станува збор за дистрибуирана апликација. На пример, доколку апликацијата се извршува дистрибуирано или користи повеќе процесорски јадра, процесот на конфигурација, инсталација и следење на процесот на извршување станува покомплициран.

Процесот на поставување на софтверот во функција може да биде поддржан од алатки кои го поедноставуваат процесот на поставување на научен софтвер. На пример, научните задачи може да бидат поставено во грид со користење на WSRF-based Grid Resource Allocation Management (GRAM) API [68]. Opal [132], исто така е алатка која обезбедува решение за поставување на научни апликации како веб услуги без притоа да се напише една линија код.

4.3 Модел за квалитет на научни апликации

4.3.1 Дефинирање на моделот за квалитет на научни апликации

Моделот на квалитет на софтвер е дефиниран преку селекција на множество на атрибути кои имаат влијание на квалитетот на научните апликации.

Изборот на атрибути е прилагоден според најважните фактори за успешност при развој на една научна апликација. Во споредба со комерцијалните апликации каде подеднаква улога има менаџментот на работата во тимот, остварувањето на профит, задоволството од клиентска страна и коректноста на

апликацијата, кај научните апликации најголем акцент се става на коректноста на софтверот.

Токму тоа е идејата и на предложениот модел за развој на научни апликации, да го подобри квалитетот преку внесување на практики на софтверското инженерство кои директно или индиректно ќе влијаат на подобрувањето на процесот на развој (особено тестирањето), а со тоа и на коректноста на софтверот.

Моделот за квалитет има за задача да го оцени квалитетот на научниот софтвер преку одредени метрики кои ќе бидат дефинирани за атрибутите вклучени во моделот. Резултатите од ова истражување се објавени во [130].

Според стандардите за квалитет IEEE Std. 1061[94] и ISO Std. 9126 [97] не постои универзално множество на атрибути кои го одредуваат квалитетот на софтверот. Во зависност од типот на проблемот и околностите треба да се изберат соодветни атрибути. Секој од избраните атрибути може да се оцени преку други атрибути од кои зависи избраниот атрибут или директно преку метрики кои можат да се изразат со број.

Моделот за евалуација на квалитетот на научни апликации прави селекција на атрибути за квалитет кои понатаму може да се разложат на други атрибути. На крајот, се избираат метрики со кои може квантитативно да се изразат селектираните атрибути.

Формално, зависноста меѓу атрибутите може да ја претставиме на следниот начин: Нека со $QA = \{A_1, A_2, ..., A_n\}$ го означиме основното (почетно) множество на атрибути за квалитет. Секој атрибут A_i од множеството QA може да зависи од друго множество атрибути $QA_i = \{A_{i_1}, .., A_{i_m}\}$, каде што m не е фиксен број. Понатаму, атрибутите од множеството QA_i може да зависат од други атрибути, итн. Атрибутите на квалитет во едно множество се независни и може да одредуваат повеќе различни атрибути.

4.3.2 Избор на атрибути и метрики за квалитет

Земајќи ги в предвид карактеристиките на научните апликации опишани низ докторската дисертација, моделот за квалитет на научни апликации го содржи следното основно множество на атрибути за квалитет: $QA = \{M, P, R\}$ каде што M се однесува на одржливост (maintainability), P означува преносливост (portability) и R доверливост (reliability).

Дефиницијата на овие атрибути според стандардот IEEE Std. 1061 е дадена во продолжение:

- Одржливост - се однесува на потребниот напор за специфични промени на софтверот. Промените може да вклучуваат корекции после грешка, подобрувања или адаптирање на софтверот за да се постигне усогласеност со барањата или околината на извршување.

- Преносливост - способност на софтверот да биде пренесен од една во друга околина.

- Доверливост - способност на софтверот да го одржува нивото на перформанси под наведени услови за одреден временски период.

Зошто моделот ги вклучува токму овие атрибути?

Изборот на овие атрибути се базира на карактеристиките на научните апликации. Имено, главната цел на научните апликации е да се справат со големо множество на податоци во разумно време и секако резултатот да биде во рамки на дозволеното отстапување.

Што се однесува до одржливоста, научните апликации треба да бидат структурирани така што ќе обезбедат можност за промени притоа без да се вложи голем напор. Оваа е важно бидејќи кога ќе се појави грешка, кодот треба да биде добро структуриран така што ќе се знае точно на кои делови од софтверот влијае грешката. Исто така, понекогаш мора да се направи промена за да се зголеми ефикасноста на извршување на апликацијата поради ограничените мемориски ресурси и време.

Преносливоста е важен аспект за квалитет кај научните апликации бидејќи често се случува повторно искористување на одредени делови од апликацијата, па дури и искористување на целиот софтвер со додавање на нови функционалности, особено доколку станува збор за апликација од ист или сличен домен. За успешна преносливост најважен фактор е независноста на софтверот (софтверските компоненти).

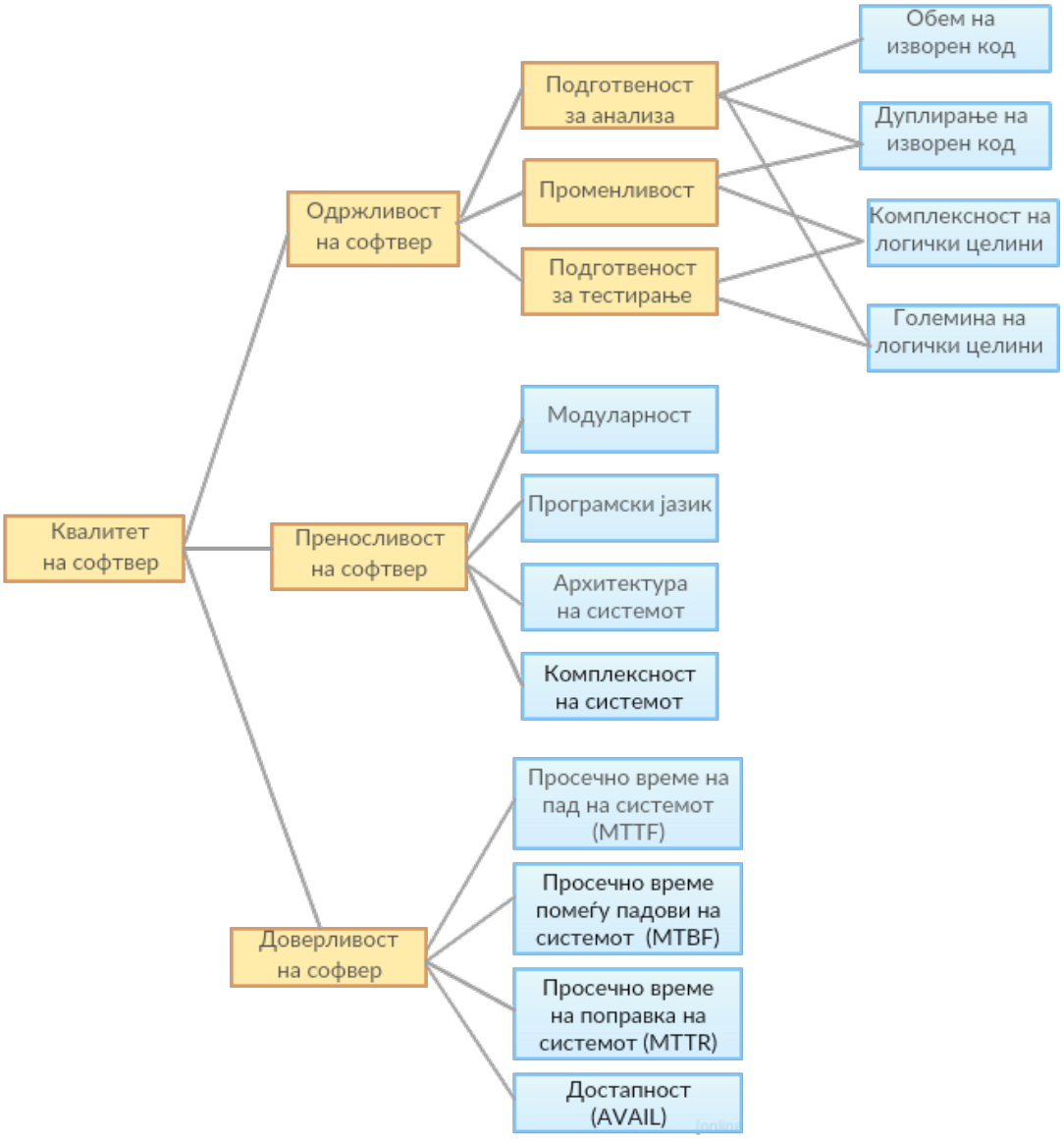
Доверливоста кај научните апликации е значајна поради тоа што софтверот треба да може се извршува подолго време без да се појават грешки. Симулациите кај научните апликации може да траат долг временски период и тоа претставува уште една причина софтверот да остане во стабилна состојба.

Според стандардот IEEE Std. 1061, се посочуваат и други атрибути за квалитет, како на пример функционалност, ефикасност и употребливост, но овие атрибути не ги вклучува моделот за квалитет на научни апликации бидејќи се поврзани со корисниците, а кај научните апликации крајните корисници се најчесто самите научници. Имено, функционалноста служи за потврда на одредени својства и функции кои ги задоволуваат наведените потреби на корисниците. Употребливоста го одредува напорот потребен за користење на софтверот од страна на корисниците. Ефикасноста вклучува и искористување на човечкиот фактор во процесот на развој, но бидејќи научната апликации е најчесто развиена од еден човек, овој фактор нема да биде значаен за проценка на квалитетот на научните апликации.

Од дефинираното основно множество на атрибути за квалитет: $QA = \{M, P, R\}$ каде што M се однесува на одржливост (maintainability), P означува преносливост (portability) и R доверливост (reliability), понатаму може да се дефинираат нови атрибути кои ги одредуваат атрибутите од множеството или пак директно да се одредат метрики.

Дополнително, пред сите метрики кои влијаат на одреден атрибут може да се стават различни тежински фактори, во зависност од она што најмногу се вреднува во софтверскиот систем. Исто така, средната вредност од сите атрибути може да го одреди квалитетот на системот.

На слика 4.3 е прикажан изборот на атрибути и метрики за оценување на квалитетот кај научен софтвер. Притоа, со жолти правоаголници се прикажани атрибутите, а со сини метриките.



Слика 4.3 – Атрибути и метрики за квалитет кај научен софтвер.

Пресметување на атрибутот одржливост

Во тој контекст, атрибутот M (одржливост - maintainability) од множеството QA зависи од повеќе атрибути: подготвеност за анализа (analysability), променливост (changeability), подготвеност за тестирање (testability) [97].

Според стандардот ISO 9126-1 подготвеноста за анализа се дефинира како способност на софтверскиот производ да биде дијагностициран со недостатоци или причини за грешки во софтверот, или пак деловите кои треба да се модифицираат да бидат идентификувани. Променливоста претставува способност на софтверскиот производ да овозможи реализација на одредена модификација. Подготвеноста за тестирање претставува способност на софтверски производ да овозможи модифицираниот софтвер да биде валидиран.

Во моделот за квалитет на научни апликации ќе бидат разгледани следните две можни сценарија за одредување на одржливоста на софтверот:

1. Одржливоста на софтверот може да се пресмета директно преку пресметување на индекс на одржливост [156].

2. Одржливоста на софтверот може да се пресмета со помош на одредување на зависност меѓу споменатите атрибути од кои зависи одржливоста (подготвеност за анализа (analysability), променливост (changeability), подготвеност за тестирање (testability)) и особините на изворниот код преточени во метрики[86].

Сценарио 1: Индексот на одржливост е дефиниран на следниот начин:

$$MI = 171 - 5.2 * ln(V) - 0.23 * (G) - 16.2 * ln(LOC) \tag{4.1}$$

каде што V е мерка за обем дефинирана од Халстед (Halstead Volume) [79], G означува цикломатска сложеност (cyclomatic complexity) [147], и LOC е бројот на линии код.

Според Халстед, обемот на еден код се пресметува на следниот начин:

$$V = N * log_2(n) \tag{4.2}$$

каде што N е вкупниот број на операции и операнди во кодот, а n е бројот на различни операции и операнди во кодот.

Цикломатската сложеност се однесува на графот на контрола на проток генериран од изворниот код и служи за дефинирање на минималниот број на случаи за тестирање на модул. Се пресметува на следниот начин:

$$G = E - N + P \tag{4.3}$$

каде што E е бројот на ребра во графот, N е бројот на темиња и P е број на компоненти на сврзаност во графот. Доколку графот е сврзан $P = 2$. Поедноставната варијанта на оваа формула е $G = D + 1$ каде што D го означува бројот на точки на одлука во графот.

Изведената формула за пресметување на индексот на одржливост која се користи од Microsoft Visual Studio (од верзијата v2008) користи скала од 0-100 и ја има следната форма :

$$MI = MAX(0, (171 - 5.2 * ln(V) - 0.23 * (G) - 16.2 * ln(LOC)) * 100/171) \tag{4.4}$$

Изведената формула за пресметување на индексот на одржливост од Институтот за софтверско инженерство ја има следната форма:

$$MI = 171 - 5.2 * log_2(V) - 0.23 * G - 16.2 * log_2(LOC) \tag{4.5}$$

Според формулата имплементирана во Visual Studio, доколку индексот на одржливост е помал од 10, одржливоста е мала, а над 20 одржливоста над софтверот е голема.

Според [47], границите се 65 и 85, наместо 10 и 20.

Сценарио 2: Во ова сценарио следниве карактеристики на кодот се директно поврзани со одржливоста на софтверот:

- Обемот на изворниот код влијае негативно на подготвеноста за анализа на софтверот. Кога станува збор за различни програмски јазици може да се направи нормализација на продуктивноста на јазиците [105]. Метрика за обемот на изворниот код може да биде бројот на линии на код со тоа што се применуваат следните правила: се отстрануваат сите празни линии и линии со коментари; се отстрануваат сите линии чија што должина е помала од два карактери. Друга опција е да се пресмета MY (man years) и според тоа да се одреди категоријата на софтверот. Пресметката може да се направи според табелата за продуктивност на програмски јазици [105]. Во оваа табела се прикажува колку линии на изворен код (LOC) одговараат на функциски поени (FP), и колку FP месечно еден програмер може да произведе кога користи одреден програмски јазик. Според тоа, доколку бројот на FP добиени од линиите код се подели со бројот на FP по програмер помножен по 12 (за цела година) ќе се добие MY. Една можна поделба на категории би била следнава: 0-8 MY, 8-30 MY, 30-80 MY, 80-160 MY, >160 MY, каде што 0-8 MY е најдобро, а >160 најлошо [86].

- Комплексноста на логички целини (units) на изворниот код влијаат негативно на променливоста на системот и подготвеноста за тестирање. Комплексноста на софтверските целини (units) може да се пресмета со користење на циколматска комплексност според формулата за цикломатска комплексност дефинирана во првото сценарио. Според Институтот за софтверско инженерство [159], постојат четири категории на ризик на единиците на софтвер според цикломатска комплексност: (1-10) - едноставна, без многу ризик; (11-20) - малку комплексна, умерен ризик; (21-50) комплексна, висок ризик; (>50) - без можност за тестирање, многу висок ризик. Според оваа поделба, може да се одреди процентот на линии код за секоја категорија.

- Степенот на дуплирање на изворниот код (клонирање на код) влијае негативно на подготвеноста за анализа и променливоста. За дуплиран код се сметаат сите блокови од код од најмалку 6 линии кои се повторуваат. Може да се пресметаат како процент на линии код во однос на целиот изворен код. Доколку процентот на линии код кој што е клониран е помал од 10% тогаш кодот е прифатлив.

- Големината на логичките целини од софтверот (units) влијае негативно врз нивната подготвеност за анализа и подготвеност за тестирање, а со тоа и врз подготвеноста за тестирање на целиот код. Метриката за големината на логички целини е повторно бројот на линии код (LOC).

Пресметување на атрибутот преносливост

Преносливоста на софтверот зависи од програмскиот јазик, библиотеките, зависноста од системските повици, и претпоставките за основниот хардвер, вклучувајќи екран, простор за складирање, достапност до меморија и дозволи

за пристап. Преносливоста на софтверот зависи од следниве метрики: модуларност (Mo), програмски јазик (Pl), проценка за архитектурата на системот (Oa) и комплексноста на системот (Co) [104].

Модуларноста на системот се пресметува според следната формула:

$$Mo = Mn * 0.5 + 1/Ms * 0.5 \tag{4.6}$$

.

Mn е бројот на модули во софтверот, а Ms е просечна големина на модул (изразена во LOC).

Проценката на архитектура на системот Oa се пресметува на следниот начин: доколку се користи програмски јазик кој е независен од платформата се добива вредност 1, доколку не, вредноста се добива од следниве три параметри кои учествуваат со по 1/3 во финалната оценка и се вреднуваат од 1 до 5 (1 значи најдобро): колку е одделен кодот зависен од платформа од кодот независен од платформа, дали се користат стандардизирани и широко достапни апликациски програмибилни интерфесји (APIs) и стандардни модели за приказ на податоци и дали се користат библиотеки зависни од платформата.

Вредноста за програмскиот јазик Pl се добива на следниот начин: доколку програмскиот јазик е еден од следниве јазици: Java™, C, C++, Python™ се вреднува со 2, а другите програмски јазици на високо ниво се вреднуваат со 1.

Комплексноста на системот Co се пресметува според следнава формула:

$$Co = G * 0.5 + Cp * 0.5 \tag{4.7}$$

каде што G е просечна цикломатска комплексност од сите модули, а Cp е мерка за зависност меѓу модулите (coupling). Зависноста на еден модул може да се пресмета на следниот начин [165]:

$$C = 1 - \frac{1}{d_i + 2 * c_i + d_o + 2 * c_o + g_d + 2 * g_c + w + r} \tag{4.8}$$

каде што d_i е бројот на влезни податочни параметри, c_i е бројот на влезни контролни параметри. Соодветно, d_o и c_o се броевите за излезни параметри. Со g_d е означен бројот на глобални променливи кои се користат како податоци, а со g_c бројот на глобални променливи кои се користат за контрола. w е бројот на повикани модули, а r е бројот на модули во кој се повикува модулот за кој се пресметува зависноста.

Cp означува просечна зависност од сите модули.

Пресметување на атрибутот доверливост

Според американскиот национален институт за стандарди (ANSI) [178], софтверската доверливост претставува веројатност да не се случи грешка во софтверот за одреден временски период во одредена околина на извршување. Според тоа, може да се дефинираат метрики кои ја одредуваат доверливоста на системот. Следниве метрики ја определуваат доверливоста на софтверски систем: просечно време на пад на системот (MTTF), просечно време помеѓу

падови на системот (MTBF), честота на пад на системот (ROCOF), просечно време на поправка на системот (MTTR), достапност (AVAIL) [109].

Просечното време на пад на системот (MTTF) се дефинира просечно време помеѓу два последователни падови на софтверски системот.

Просечно време на поправка на системот (MTTR) се дефинира како просечно време потребно за поправка на системот после пад.

Просечно време помеѓу падови на системот (MTBF) се дефинира како:

$$MTBF = MTTF + MTTR \tag{4.9}$$

Честота на пад на системот (ROCOF) се дефинира како број на падови на софтверот во одредена временска единица.

Достапност (availaibility) е атрибут кој ја одредува доверливоста на системот и може да се пресмета како:

$$AVAIL = \frac{MTBF}{MTBF + MTTR} \tag{4.10}$$

Глава 5

Примена на рамката за развој на научни апликации

Практичната примена на рамката за квалитет е прикажана преку две студии на случај: апликација за решавање на еднодимензионална и дводимензионална Шредингерова равенка со методот со репрезентација на дискретни променливи, а другата пресметува пулс и респираторна стапка од ЕКГ сигнал. Првата апликација го следи предложениот модел за развој на научни апликации, а другата не користи специфичен модел, туку се развива ад-хок. Квалитетот на апликациите се евалуира преку компаративна анализа направена според предложениот модел за квалитет. Резултатите од ова истражување се објавени во [129] и [130].

5.1 Практична примена на моделот за развој на научни апликации

Практичната примена за користење на предложената рамка за развој на научни апликации е прикажана преку развој на апликација која користи методот на претставување со дискретни променливи за решавање на еднодимензионална и дводимензионална Шредингерова равенка. Процесот на развој на апликацијата се заснова на моделот на развој опишан во поглавјето 4.2.2.

5.1.1 Опис на научниот проблем

Методите со репрезентација на дискретни променливи (DVR) се користат во многу научни области како што се компјутерска физика, хемија, квантна динамика, итн.

DVR методите имаат широка примена бидејќи решаваат ефикасно нумерички проблеми од квантната динамика. Овие методи ја поедноставуваат пресметката на матричните елементи на операторот на кинетичката енергија, како и матричните елементи на операторот на потенцијалната енергија во рамките на DVR. Во случај на мултидимензионални системи, Хамилтоновата матрица е ретка матрица и операцијата на Хамилтонијанот над вектор е брза кога се пресметува производот на DVR [141].

Шредингеровата равенка е парцијална диференцијална равенка која се користи за опишување на динамиката на системот на атомично и молекуларно ниво. Временски независната (стационарна) Шредингерова равенка се запишува на следниот начин:

$$H\Psi = E \cdot \Psi \tag{5.1}$$

каде H е операторот на Хамилтон, Ψ е бранова функција на квантниот систем и E е енергијата во состојбата . Временски зависната Шредингерова равенка може да се претстави на следниот начин:

$$i \cdot \hbar \frac{\partial \Psi}{\partial t} = H\Psi \tag{5.2}$$

каде $\hbar$ е Планкова константа поделена со 2π, а $\frac{\partial}{\partial t}$ е парцијален извод по времето t .

Еден од начините за решавање на Шредингеровата равенка со користење на матрична репрезентација е методот на претставување со дискретни променливи.

Матричните елементи на диференцијалните оператори се пресметани точно, а оние кои се дијагонални во координатна репрезентација како што е операторот на потенцијална енергија се пресметани приближно со користење точност на Гаусова квадратура. Гаусовата квадратура дава најприближна апроксимација на интеграл за даден број на точки и тежини и со тоа помага во точноста и ефикасноста на DVR методот [196].

5.1.2 Идентификација на модули на системот

На почетокот беа идентификувани модулите дадени во продолжение. Останатите модули кои беа додавани во текот на развојот се прикажани во прилогот А.

1. Модул за множење на две квадратни матрици (влез - 2 квадратни матрици и бројот на редици/колони на матрицата, излез – матрица)

2. Модул за креирање на дијагонална матрица (влез – низа и број на редици/ колони на матрицата, излез – матрица)

3. Модул за множење на скалар со квадратна матрица (влез – матрица, скалар и број на редици/колони на матрицата, излез – матрица)

4. Модул за креирање на единечна матрица (влез – број на редици/ колони на матрицата, излез – матрица)

5. Модул за собирање на две матрици (влез - 2 матрици и бројот на редици/ колони на матрицата, излез – матрица)

6. Модул за креирање на транспонирана матрица (влез – број на редици/ колони на матрицата, излез – матрица)

7. Структура за една редица од влезната датотека – податоци: три броја

8. Модул за читање на податоци од датотека – (влез патеката на датотеката, излез – низа од структури за една редица од влезната датотека)

9. Модул за сортирање на редици во датотека (влез - низа од структури за една редица од влезната датотека, излез - низа од структури за една редица од влезната датотека)

10. Модул за пресметување на сопствени вредности (влез – матрица и број на редици/колони на матрицата, излез – низа)

11. Модул („thcheby") за пресметување на низи на вредности за x и y, матрици за трансформација во репрезентација со база во конечно димензионален простор (FBR), (влез – број на точки за вредности на x, број на точки за вредности на y, минимални и максимални вредности за x и y, излез – низа за вредности на x и низа за вредности на y, низа за вредности на x во FBR и низа за вредности на y во FBR, матрици за трансформација на вредностите за x и y од FBR во DVR. Вредностите за y се потребни само за решавање на дводимензионална Шредингерова равенка)

12. Модул за наоѓање на локалниот минимум на дадена функција до дадена точка - (Влез: точка, функција, излез: локален минимум)

13. Модул за интерполација (Метод на Хук) - (влез: почетна точка, крајна точка, максимална вредност на итерација, излез: интерполирана вредност)

14. Модул за трансформација на влезната датотека на точки и вредности на функцијата во тие точки - (влез: датотека, излез: трансформирана датотека)

15. Модул за решавање на еднодимензионална и дводимензионална Шредингерова равенка 1D и 2D Schrödinger преку користење метод на претставување со дискретни променливи (DVR) (влез: 1D или 2D мрежа од точки, максимални и минимални вредности на x и y, број на точки во DVR , функции за маса и интерполација; излез: вибрациони бранови функции и вибрациони фреквенции на транзиција.

5.1.3 Дизајн на системот

Апликацијата може да се изврши на машина со еден процесор на било кој оперативен систем. Апликацијата е напишана во програмскиот јазик C и за повторно компајлирање потребен е компјалер за програмскиот јазик C (gcc компајлер). За било какви модификации, потребен е уредувач на текст.

Од апликацијата се генерира датотека која корисникот може да ја вклучи и да ги користи сите дефинирани методи. За решавање на еднодимензионална или дводимензионална Шредингерова равенка, треба да се внесат влезни параметри и патеката до датотеката со податоци (вредности за потенцијалната енегрија пресметани на двозимензионален грид од точки или еднодимензионален грид од точки прочитан од датотека, бројот на DVR точки, минималните

и максималните вредности x за и y, функција за интерполација и маса), како што е наведено во секцијата 5.1.2.

Резултатите се дадени како стандарден излез. За генерирање и извршување на тестовите се користи рамката „CuTest" [101].

При развој на апликацијата мора да се земат в предвид следните ограничувања (нефункционални барања):

1. Комплексноста на алгоритмот за сортирање на низа треба да биде $< O(n^2)$;
2. Потребно е да се користи динамичка алокација на меморија;
3. Треба да се прави ослободување на меморија на променливите секогаш кога тие веќе не се употребуваат.

5.1.4 Спецификација на модули на системот

Детална спецификација на барања за модул

Спецификацијата на барањата на системот е направена според темплејтот предложен во глава 4.2.3. Како примери, во продолжение се дадени спецификации на две барања кои се однесуваат за два различни модули.

Табела 5.1 – Спецификација на барање на модул за креирање на дијагонална матрица.

ИД	02
Верзија	1.0
Име	Модул за креирање на дијагонална матрица
Опис	Модул за креирање дијагонална квадратна матрица од низа броеви. Ако низата е означена со **A**, а матрицата со **M**, тогаш **M**[**i**][**i**] = **A**[**i**]. Елементите кои не лежат на главната дијагонала се 0. Резултатот е матрицата **M**.
Историја	/

Дизјан на модул

За развивање на модулите на оваа апликација не се јавува потреба за користење на одредени шаблони и техники за програмирање. Единствено, за сортирањето на редиците во датотеката во модулот 9 се користи методот на брзо сортирање (quick sort).

Табела 5.2 – Спецификација на барање на модул за генерирање на низи за x и y вредности, низи за x и y вредности во FBR и матрици на трансформација на x и y вредностите од FBR во DVR („thcheby").

ИД	11
Верзија	1.0
Име	Модул за генерирање на низи за x и y вредности, низи за x и y вредности во FBR и матрици на трансформација на x и y вредностите од FBR во DVR („thcheby").
Опис	Модулот има 5 влезни аргументи: nx (број на вредности за x - цел број), $xmin$ (минимална вредност на x- реален број), $xmax$ (максимална вредност на x – реален број), ny (број на y вредности - цел број), $ymin$ (минимална вредност на y - реален број), $ymax$ (максимална вредност на y - реален број). Првин се пресметуваат разликите помеѓу максималната и минималната вредност на x ($deltax$) и соодветно за y ($deltay$). Следно, низата од x вредности ($ptsx$) и низата од y вредности ($ptsy$) се генерираат со повикување на функциите: $make\_array\_ptsxy$ (nx,$nx+1$,$xmin$,$deltax$) за вредностите на x и $make\_array\_ptsxy$ (ny,$ny+1$,$ymin$,$deltay$) за вредностите на y. Потоа, елементите на низите за x вредности ($fbrtx$) и за y вредности($fbrty$) во FBR се пресметуваат со повикување на функцијата $make\_array\_fbrtxy(deltax,nx)$ за вредностите за x и $make\_array\_fbrtxy(deltay,ny)$ за вредностите на y. На крајот се креираат матриците на трансформација Tx и Ty со повикување на функциите $make\_matrix\_Txy(nx,nx+1)$ за вредностите на x и $make\_matrix\_Txy(ny,ny+1)$ за вредностите на y.
Историја	/

Дизајн на случаи за тестирање

Создавањето тест случаи пред пишување на кодот е корисно за идентификување на потребните влезови, очекуваните излези и условите. Ова е фазата кога се генерираат само текстуалните документи, а подоцна, во фазата на тестирање, се користат за генерирање и извршување на тестовите. Пример за спецификација за тест случај е посочен во табелите 5.3 и 5.4. По завршување на фазата на тестирање за модулот, статусот на тестот се менува во успешен или неуспешен (PASS/FAIL).

Табела 5.3 – Пример за дефиниција на тест случај – Модул за креирање на дијагонална матрица

ИД	02			
Име	Дијагонална матрица			
ИД на барање	2			
Предуслови	На влез треба да се приложи низа од n реални броеви и n мора да биде цел број.			
Чекор	Акција	Очекуван резултат	Статус (Успешен/Неуспешен)	Коментари
1	Иницијализација на 2D низа $\mathbf{M}$ со големина $n \times n$	Алоцирана меморија за $\mathbf{M}$ со големина $n \times n$. Елементите се реални броеви	Успешен	Се користи динамичка алокација
2	Додавање елементи на матрицата $\mathbf{M}$	Елементи на низата $\mathbf{A}$ се поставени на главната дијагонала на матрицата, другите елементи се 0. $\mathbf{M[i][i]} = \mathbf{A[i]}$, $\mathbf{M[i][j]} = \mathbf{0}$, $i! = j$	Успешен	/

Табела 5.4 – Пример за дефиниција на тест случај – Модул за генерирање на низи за x и y вредности, низи за x и y вредности во FBR и матрици на трансформација на x и y вредностите од FBR во DVR („thcheby").

ИД	11			
Име	Генерирање на низи за x и y вредности, низи за x и y вредности во FBR и матрици на трансформација на x и y вредностите од FBR во DVR. .			
ИД на барање	11			
Предуслови	Бројот на вредности за x и y мора да биде познат (nx и ny) Минималните и максималните вредности за x и y мора да бидат иницијализирани ($xmin$,$xmax$,$ymin$,$ymax$).			
Чекор	Акција	Очекуван резултат	Pass/Fail (Статус)	Коментари
1	Пресметка на разликата помеѓу максималната и минималната вредност на x	$deltax = xmax - xmin$ Елементите се од тип double	Успешен	/
2	Пресметка на разликата помеѓу максималната и минималната вредност на y	$deltay = ymax - ymin$ Елементите се од тип double	Успешен	/
3	Генерирање на елементите од низата $ptsx$	$ptsx[i] = ((i+1)*$ $deltax * 1.0)/(nx+1)+$ $xmin$ Елементите се од тип double	Успешен	Резултатите се тестирани со тестирање на функцијата $make\_array\_ptsxy$ (nx,$nx+1$,$xmin$,$deltax$)
4	Генерирање на елементите од низата $ptsy$	$ptsy[i] = ((i+1)*$ $deltay * 1.0)/(ny+1)+$ $ymin$ Елементите се од тип double	Успешен	Резултатите се тестирани со тестирање на функцијата $make\_array\_ptsxy$ (ny,$ny+1$,$ymin$,$deltay$)
5	Генерирање на елементите од низата $fbrtx$	$fbrtx[i] = ((\Pi \times (i+1))$ $/deltax)^2$ Елементите се од тип double	Успешен	Резултатите се тестирани со тестирање на функцијата $make\_array\_fbrtxy(deltax, nx)$
6	Генерирање на елементите од низата $fbrty$	$fbrty[i] = ((\Pi \times (i+1))$ $/deltay)^2$ Елементите се од тип double	Успешен	Резултатите се тестирани со тестирање на функцијата $make\_array\_fbrtxy(deltay, ny)$
7	Генерирање на елементите на матрицата Tx	$Tx[i][j] = \sin((i+1) \times (j+1) \times$ $\sqrt{2.0/(nx+1)} \times \Pi/(nx+1))$ Елементите се од тип double	Успешен	Резултатите се тестирани со тестирање на функцијата $make\_matrix\_Txy(nx, nx+1)$
8	Генерирање на елементите на матрицата Ty	$Ty[i][j] = \sin((i+1) \times (j+1) \times$ $\sqrt{2.0/(ny+1)} \times \Pi/(ny+1))$	Успешен	Резултатите се тестирани со тестирање на функцијата $make\_matrix\_Txy(ny, ny+1)$

Имплементација

Низите се дефинирани со користење на покажувачи и секогаш кога е можно се прави ослободување на меморија. Пресметката на сопствените вредности се прави користејќи ја GNU Scientific (GSL) библиотеката. Комплексноста на алгоритмот е $O(n^2)$. Сортирањето на редиците во датотеката се врши користејќи го методот на брзо сортирање (quick sort). Исто така, низ целиот код се додадени коментари кои ги опишуваат модулите, променливите, јамките и наредбите. Променливите и методите имаат јасни имиња.

Кодот на апликацијата е поделен на модули. Секој модул е независен дел од кодот кој се имплементира по спецификација на тест случаите за истиот модул. Се користат концизни и описни имиња за променливите и методите. Целта е запишаните модули да се колку може повеќе независни. Исто така, се креира библиотека која што може да биде вклучена во било кој друг проект и методите можат да бидат повикани со спецификација на бараните аргументи на методот. Дополнително, прикажани се и коментари за опис на состојбата на променливите.

Тестирање

Тестирањето е извршено со користење на CuTest системот кој што е дизајниран за пишување, администрирање и извршување на тестови во програмскиот јазик C. Тестирањето на некој тест случај е успешно завршено доколку сите барања и услови за тој тест случај се исполнети. Пример за функција за тестирање е дадена на слика 5.1.

Евалуација

Се врши споредба на резултатите од извршените тестови со резултатите добиени со извршување на програмскиот код напишан во Mathematica од Универзитетот Upssala [33] [32]. Исто така, приближувањето на резултатите е евалуирано преку зголемување на бројот на базисни функции или преку зголемување на густината на мрежата. Грешките кои беа најдени во текот на развојот на некој модул се коригира веднаш и повторно беше извршено тестирањето на тој модул. На крајот на развојот, статусот на сите тест случаи беше: успешен. Исто така, во предвид беа земени и нефункционалните барања.

5.1.5 Интеграциско тестирање

На крајот на развојот беше направено интеграциско тестирање. Интеграциското тестирање се изврши преку тестирање на модулот DVR2Dv1 со кој се добива решението на Шредингеровата равенка. Во овој модул директно или индиректно се повикуваат сите останати модули во апликацијата. Начинот на тестирање на овој модул беше ист како и за останатите модули.

```
void Test(CuTest *cutest)
{

    struct return_objects result=thcheby(10,1,5,10,2,6);
    double *actualptsx=result.ptsx;
    double *actualfbrtx=result.fbrtx;
    double **actualTx=result.Tx;
    int i,j;
    double *expectedptsx=malloc(10*sizeof(double));
    double *expectedfbrtx=malloc(10*sizeof(double));
    double **expectedTx=malloc(10*sizeof(double));

    for(i=0; i<10; i++)
    {
        expectedptsx[i]=((i+1)*(5-1)*1.0)/11+1;
        expectedfbrtx[i]=square(((i+1)*M_PI)/(5-1));
        CuAssertTrue(tc,abs(expectedptsx[i]-actualptsx[i])
            <0.0000001);
        CuAssertTrue(tc,abs(expectedfbrtx[i]-actualfbrtx[i])
            <0.0000001);
        expectedTx[i] = malloc(10*sizeof(double));

         for(j=0; j<10; j++)
         {
           expectedTx[i][j]=sqrt(2.0/11)*sin((i+1)*(j+1)*
           M_PI/11);
           CuAssertTrue(tc,abs(expectedTx[i][j]-
           actualTx[i][j])<0.0000001);
          }
     }
}
```

Слика 5.1 – Функција за тестирање на модулот „thcheby”

5.1.6 Поставување на системот во функција

По завршеното интеграциско тестирање беше креирана библиотека што овозможува повикување на функциите во друг софтвер напишан во програмскиот јазик C. Методите се достапни преку вклучување на библиотеката во друга апликација и преку повикување на методите со соодветните влезни параметри.

5.2 Компаративна анализа на квалитет на научни апликации

Во ова поглавје се врши евалуација на квалитетот на две научни апликации користејќи го моделот за квалитет предложен во глава 4.3.

Евалуацијата на квалитет на апликациите се врши преку споредување на атрибути и метрики значајни за научните апликации, опишани во моделот на квалитет. Како тест случи се земени две апликации :

1. Апликација за решавање на еднодимензионална и дводимензионална Шредингерова равенка (A1);

2. Апликација за пресметување на пулс и респираторна стапка од ЕКГ сигнал (A2).

И двете апликации се напишани во програмскиот јазик C. A1 има 1074 линии код организиран во 48 логички единици (units, модули), а A2 1148 организирана во 26 логички единици. A1 е развиена според предложениот модел за развој на научни апликации, а A2 е развиена без користење на одреден модел за развој на софтвер, почнувајќи директно од фазата на програмирање.

Главните атрибути кои се оценуваат според предложениот модел се: одржливост, преносливост и доверливост. Атрибутите и метриките со кои се определува квалитетот на научна апликација се дефинирани на слика 4.3.

5.2.1 Пресметка на атрибутот одржливост

За пресметување на одржливоста на софтвер во глава 4.3.2 се дефинирани две сценарија. Пресметувањето на индексот на одржливост е направено според формулата 4.5. За пресметување на метриката: обем на софтвер според Халстед (V) се користи алатката Halstead Metrics Tool [1]. Резултатите од пресметувањето на индексот за одржливост MI за двете апликации се прикажани во табела 5.5. Со G е означена цикломатската комплексност, а со LOC бројот на линии код.

Табела 5.5 – Индекс на одржливост

	LOC	V	G	MI
A1	1074	27081.28	160	62.05
A2	1148	30496.80	169	59.24

Иако резултатите се во корист на A1, бидејќи индексот на одржливост го вклучува бројот на линии код директно како метрика, сметаме дека не е најрелевантен фактор за мерење на квалитет на одредена апликација.

Според второто сценарио, одржливоста на софтверот зависи од следните атрибути: подготвеност за анализа на софтверот, променливост на софтверот и подготвеност за тестирање.

Подготвеноста на анализа на софтверот може да се претстави со следните метрики:

- Обем на изворен код - се пресметува во MY(man years) и влијае негативно на подготвеноста на анализа на софтверот. Пресметката за обемот на изворниот код е направена според табелите [105]. Функциските поени (FP) се добиени кога бројот LOC ќе се подели со 128 бидејќи и кај двете апликации се користи програмскиот јазик C. Обемот на кодот изразен во MY се добива кога FP ќе се подели со бројот на FP месечно по програмер помножено со 12 месеци.

- Дуплирање на код - се пресметува во % LOC и влијае негативно на подготвеноста на анализа на софтверот. За откривање на бројот на линии кои се дуплираат се користи алатката PMD [95];

- Големина на логички целини - се пресметува како просечна големина (LOC) на логички единици (units) на кодот и влијае негативно на подготвеноста на анализа на софтверот.

Резултатите за атрибутот подготвеност за анализа се прикажани во табелата 5.6. Бидејќи сите три метрики влијаат обратнопропорционално на подготвеноста за анализа, јасно може да се заклучи дека софтверот A1 има подобри резултати. Во пресметката, сите три метрики учествуваат подеднакво.

Табела 5.6 – Атрибут: Подготвеност за анализа

	Обем (MY)	Дуплирање на LOC (%)	Просечна големина на логички единици код (LOC)	Подготвеност за анализа
A1	0.08	5.59	16.48	7.38
A2	0.09	17.25	44.12	20.48

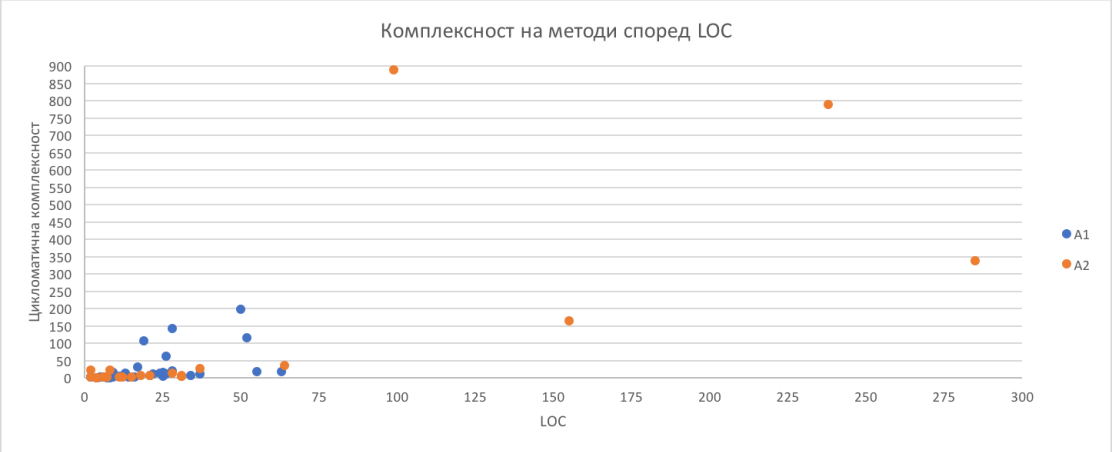
Променливоста на софтверот зависи од % на дуплиран код и од просечната комплексност на логичките целини во кодот. И двете метрики влијаат обратнопропорционално на променливоста на софтверот.

Пресметките според категориите за ризик според цикломатската комплексност предложени во глава 4.3.2, се прикажани во табела 5.7.

Табела 5.7 – Процент на изворен код според категории за ризик на цикломатската комплексност

Комплексност на модули	A1 (во %)	A2 (во %)
1-10	27.65	20.12
11-20	25.51	2.44
21-50	4.19	9.67
>50	16.29	67.68

Цикломатската комплексност на модулите во однос на бројот на линиите на модул на апликациите A1 и A2 е прикажана на слика 5.2. Може да се забележи дека апликацијата A2 има три модули со комплексност поголема од 250 што влијае негативно на комплексноста на целата апликација.



Слика 5.2 – Цикломатска комплексност на модулите во однос на бројот на линии код на модул за апликациите A1 и A2

За пресметување на променливоста на кодот се користи просечна комплексност по модул и процентот на дуплиран код. Во пресметката, двете метрики учествуваат подеднакво. Резултатите за атрибутот променливост на код се прикажани во табела 5.8. Бидејќи двете метрики влијаат обратнопропорционално на можноста за промена на кодот, може да се заклучи дека софтверот A1 има подобри резултати.

Табела 5.8 – Атрибут: можност за промена

	Дуплирање на LOC %	Просечна комплексност на логички единици	Променливост
A1	5.59	19.02	12.30
A2	17.25	90.77	54.01

Атрибутот подготвеност за тестирање зависи од просечната комплексност на логичките целини и просечната големина на логичките целини. Во пресметката, двете метрики учествуваат подеднакво. Резултатите за атрибутот подготвеност за тестирање се прикажани во табела 5.9. Бидејќи двете метрики влијаат обратнопропорционално на подготвеноста за тестирање на кодот, може да се заклучи дека софтверот A1 има подобри резултати.

Табела 5.9 – Атрибут: подготвеност за тестирање

	Просечна големина на логички единици	Просечна комплексност на логички единици	Подготвеност за тестирање
A1	16.48	19.02	17.75
A2	44.12	90.77	67.44

5.2.2 Пресметка на атрибутот преносливост

Атрибутот преносливост на софтвер се пресметува преку четири метрики: модуларност, програмски јазик, архитектура на системот и комплексност на системот.

Модуларноста на системот зависи од бројот на модули и од просечната големина на модулите (LOC) и се пресметува според формулата 4.6. Бројот на модули влијае позитивно на модуларноста на системот, а просечната големина на модулите негативно. Двете метрики учествуваат подеднакво. Резултатите за модуларноста се прикажани во табела 5.10. Од табелата може да се заклучи дека софтверот A1 има подобри резултати.

Табела 5.10 – Атрибут: модуларност

	Просечна големина на логички единици	Број на модули	Модуларност
A1	16.48	48	24.03
A2	44.12	26	13.011

Бидејќи се користи програмски јазик кој е независен од платформата, архитектурата на системот се оценува со вредност 1. За програмскиот јазик се добива вредност 2 кај двете апликации.

Комплексноста на системот се пресметува според формулата 4.7. Просечната циколоматска комплекснот по модул и просечната зависност од други модули по модул влијаат обратнопропорционално на комплексноста на системот. Резултатите за модуларноста се прикажани во табела 5.11. Од табелата може да се заклучи дека софтверот A1 има подобри резултати во однос на комплексноста на системот.

Табела 5.11 – Атрибут: комплексност на системот

	Просечна комплексност на логички единици	Просечна зависност од други модули	Комплексност на системот
A1	19.02	0.82	9.92
A2	90.77	0.77	45.77

5.2.3 Пресметка на атрибутот доверливост на софтвер

Доверливоста е определена според следниве метрики: просечно време на пад на системот (MTTF), просечно време помеѓу падови на системот (MTBF), просечно време на поправка на системот (MTTR) и достапност (AVAIL). Само метриката MMTR влијае обратнопропорционално на доверливоста.

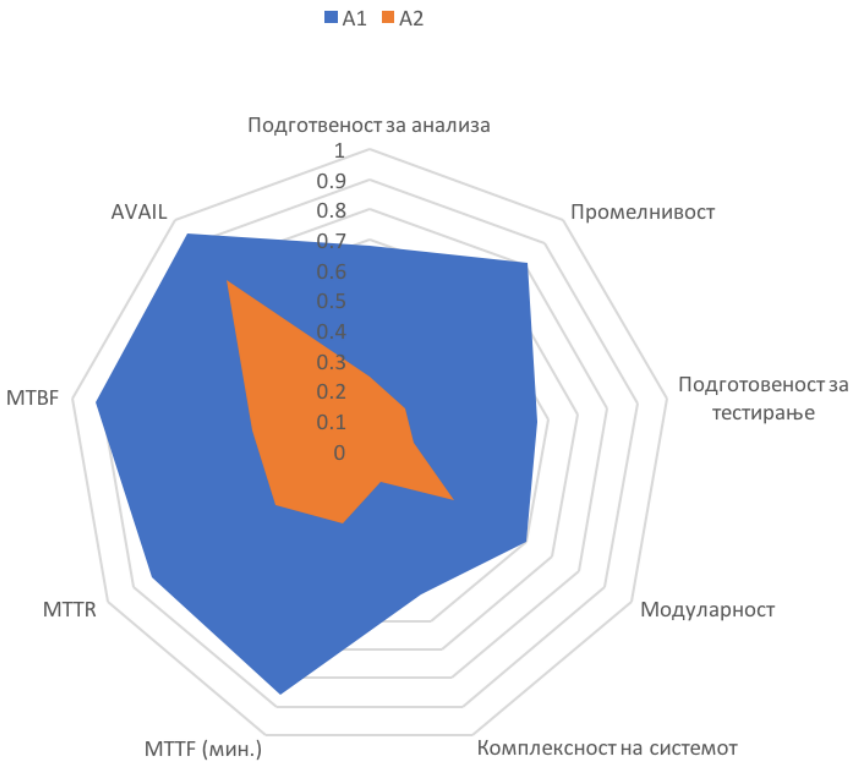
Метриките се пресметани според формулите прикажани во главата 4.3.2. Резултатите за апликацијата A1 се добиени од вкупно 173 минути, а резултатите за апликацијата A2 од 172 минути. Истите се прикажани во табела 5.12. За апликацијата A1 се добиваат подобри резултати за сите метрики.

Споредбата на сите метрики кои го одредуваат квалитетот на научните апликации A1 и A2 е прикажана на слика 5.3. Вредностите на метриките се

Табела 5.12 – Атрибут: доверливост

	MTTF (мин.)	MTTR (мин.)	MTBF (мин.)	AVAIL
A1	86	6	92	0.94
A2	25.37	14	39.37	0.74

скалирани од 0 до 1. Дополнително, за оние метрики кои влијаат обратнопропорционално на квалитетот, се земаат в предвид нивните реципрочни вредности. Метриките не се зависни меѓу себе. Од сликата може да се воочи дека апликацијата A1 која го следи предложениот модел за развој на научни апликации има подобри вредности за сите метрики кои го определуваат квалитетот на научниот софтвер во однос на апликацијата A2. Резултатите укажуваат на тоа дека користењето на моделот за развој на научни апликации го подобрува квалитетот на апликациите. Деталните резултати за евалуација на метриките за секој модул од апликациите A1 и A2 се прикажани во додатокот А.



Слика 5.3 – Споредба на метриките за квалитет кај апликациите A1 и A2

Глава 6

Заклучок

Со ова истражување се воочуваат проблемите со кои се соочуваат научниците во текот на процесот на развој на научна апликација. Истражувањето помогна за идентификување и препорака на практики од софтверското инженерство кои можат да се применат при развој на научен софтвер.

Разликата на начинот на развој на комерцијален и научен софтвер води кон разлики во концептот на евалуација на квалитетот. Сѐ уште има има многу отворени прашања во врска со квалитетот на научните софтвери, примарно во однос на критериумите за квалитет и практиките кои треба да се применат во таа насока.

Во таа насока, за да се даде одговор на овие прашања, во оваа докторска дисертација се предлага рамка за развој на научни апликации. Рамката опфаќа модел за развој на научни апликации и модел за квалитет за научни апликации.

Целта на предложениот модел за развој на научни апликации е да се променат практиките за развој на научниците со тоа што ќе се посвети повеќе внимание на оние фази на развој кои воопшто не се практикуваат од страна на научниците, како и да се подобри квалитетот на процесот на развој на софтверот. Предложениот модел вклучува особини од повеќе процесни модели (итеративен, инкрементален, агилен, модел на прототипи), промена во постоечките фази (различна евалуација) и дополнително при развој на секој модул додава нови чекори: дизајн на модул, дизајнирање на случаи за тестирање пред имплементација на модул.

Моделот за квалитет вклучува и квантитативна евалуација на атрибути кои се вреднуваат во секоја од фазите на развој на научните апликации. Моделот на квалитет се дефинира преку селекција на множество на атрибути кои имаат влијание на квалитетот на научните апликации. За евалуација на научните апликации се избираат следниве главни атрибути, кои понатаму се претставуваат преку други атрибути и метрики: одржливост (maintainability), преносливост (portability) и доверливост (reliability). При дефинирање на моделот за квалитет на научни апликации, се земаат в предвид веќе прифатени стандарди за квалитет.

Практичната примена на рамката за квалитет е прикажана преку две студии на случај: апликација за решавање на еднодимензионална и дводимензионална Шредингерова равенка со методот со репрезентација на дискретни

променливи, а другата пресметува пулс и респираторна стапка од ЕКГ сигнал. Првата апликација го следи предложениот модел за развој на научни апликации, а другата не користи специфичен модел, туку се развива ад-хок. Квалитетот на апликациите се евалуира преку компаративна анализа направена според предложениот модел за квалитет.

Користењето на моделот за развој на научни апликации, предложен во оваа дисертација, при развивање на апликацијата за решавање на еднодимензионална и дводимензионална Шредингерова равенка со методот со репрезентација на дискретни променливи резултира во апликација со модуларен код, специфицирани тест случаи, автоматско тестирање и генерирање документи за гаранција на квалитет.

Евалуацијата на апликациите според моделот за квалитет на научни апликации, покажа дека апликацијата која го следи предложениот модел за развој на научни апликации има подобри вредности за сите метрики кои го определуваат квалитетот на научниот софтвер во однос на апликацијата која не го следи моделот за развој.

Во дисертацијата, исто така, се применуваат практики на софтверското инженерство кои може да се прилагодат при развојот на научните апликации. Се прикажува пример за формална спецификација научни апликации користејќи интервална темпорална логика, градење на научен работен тек со повеќе алатки, примена на програмскиот модел MapReduce за развој на на научни апликации, диструбуиран развој на научни апликации. Покрај тоа, се обработуваат и темите за регресиско тестирање и тестирање при агилен развој на софтвер. Дополнително, се предлага и план за управување со промени и контрола на верзиите на научните апликации.

Рамката ќе ги води научниците низ развојниот процес и ќе помогне да се променат сегашните практитки. Дефинираните независни модули исто така може да се користат и при развој на други апликации, особено во истата научно-истражувачка заедница. Може да се заклучи дека практиките за софтверско инженерство може да се применат и при развој на научен софтвер, но притоа поради постоечките разлики меѓу комерцијалните и научните апликации мора да се направат соодветни модификации. Според досегашните заклучоци, може да се потврдат хипотезите поставени во главата 1.2.

Идното истражување во оваа област е насочено кон подобрување на моделот на развој и дефинирање на модел за предвидување на грешки кај научните апликации. Моделот треба да учи од постоечко множество на научни апликации. Со тоа, уште повеќе, ќе се подобри квалитетот на апликациите.

Прилог А

Табели за евалуација на квалитет

Табела А.1 – Метрики за апликацијата А2

модул	комлексност	LOC	влезни параметри
void diff	27	37	1
static double skip_to_last_equal_value	14	28	2
void do_vectors	166	155	5
static void findLocalMaxima	339	285	3
static void parse_inputs	36	64	6
void findpeaks	789	238	2
void b_emxInit_real_T	2	11	3
void emxEnsureCapacity	7	21	3
void emxFree_boolean_T	3	7	1
void emxFree_int32_T	3	7	1
void emxFree_real_T	3	7	1
void emxInit_boolean_T	2	12	2
void emxInit_int32_T	2	12	2
void emxInit_real_T	2	12	2
void HRRRshort_initialize	23	2	0
void HRRRshort	888	99	4
double nanmean	7	18	1
void rt_InitInfAndNaN	22	8	1
boolean_T rtIsInf	2	2	1
boolean_T rtIsNaN	2	6	1
real_T rtGetInf	5	31	0
real32_T rtGetInfF	1	4	0
real_T rtGetMinusInf	4	31	0
real32_T rtGetMinusInfF	1	4	0
real_T rtGetNaN	7	31	0
real32_T rtGetNaNF	3	15	0

Табела А.2 – Метрики за апликацијата А2

модул	излезни параметри	глобални променливи	повикува модули
void diff	0	0	1
static double skip_to_last_equal_value	1	0	2
void do_vectors	0	0	8
static void findLocalMaxima	0	0	9
static void parse_inputs	0	0	3
void findpeaks	0	0	13
void b_emxInit_real_T	0	0	0
void emxEnsureCapacity	0	0	0
void emxFree_boolean_T	0	0	0
void emxFree_int32_T	0	0	0
void emxFree_real_T	0	0	0
void emxInit_boolean_T	0	0	0
void emxInit_int32_T	0	0	0
void emxInit_real_T	0	0	0
void HRRRshort_initialize	0	0	1
void HRRRshort	0	0	7
double nanmean	1	1	1
void rt_InitInfAndNaN	0	6	6
boolean_T rtIsInf	1	2	0
boolean_T rtIsNaN	1	0	0
real_T rtGetInf	1	0	1
real32_T rtGetInfF	1	0	0
real_T rtGetMinusInf	1	0	1
real32_T rtGetMinusInfF	1	0	0
real_T rtGetNaN	1	0	1
real32_T rtGetNaNF	1	1	0

Табела А.3 – Метрики за апликацијата А2

модул	повикан во модули	зависност од други модули
void diff	1	0.666666667
static double skip_to_last_equal_value	1	0.833333333
void do_vectors	1	0.928571429
static void findLocalMaxima	1	0.923076923
static void parse_inputs	1	0.9
void findpeaks	1	0.9375
void b_emxInit_real_T	4	0.857142857
void emxEnsureCapacity	6	0.888888889
void emxFree_boolean_T	2	0.666666667
void emxFree_int32_T	3	0.75
void emxFree_real_T	6	0.857142857
void emxInit_boolean_T	2	0.75
void emxInit_int32_T	2	0.75
void emxInit_real_T	9	0.909090909
void HRRRshort_initialize	0	0
void HRRRshort	0	0.909090909
double nanmean	1	0.8
void rt_InitInfAndNaN	1	0.928571429
boolean_T rtIsInf	3	0.857142857
boolean_T rtIsNaN	6	0.875
real_T rtGetInf	1	0.666666667
real32_T rtGetInfF	2	0.666666667
real_T rtGetMinusInf	1	0.666666667
real32_T rtGetMinusInfF	2	0.666666667
real_T rtGetNaN	1	0.666666667
real32_T rtGetNaNF	2	0.75

Табела А.4 – Метрики за апликацијата А1

модул	комлексност	LOC	влезни параметри
double * make_array_ptsxy	2	6	4
double * make_array_fbrtxy	2	6	2
double** make_matrix_Txy	4	9	2
struct return_objects thcheby	17	25	6
print_n_elements_from_array	2	5	3
double** transpose_matrix	5	12	2
double** diagonal_matrix	5	12	2
double** matrix_multiplication	5	11	3
double* divide_vector_number	2	6	3
double **multiple_element_matrix	4	9	3
double **identity_matrix	5	12	1
double ***array_matrices_multiply	15	13	2
double **matrix_addition	4	9	3
double ***array_matrices_add	16	9	0
struct file_plus_rows return_data_file	2	16	1
void swap_el	1	4	4
int compare_file_row	6	8	6
struct file_plus_rows sortarray	31	17	4
double *get_eigenvalues	143	28	2
double f	4	25	4
double best_nearby	14	24	6
int hooke	19	55	8
double ** interpolation	7	34	4
double * prepare_PES_array	199	50	0
void DVR2Dv1	107	19	7
gsl_eigen_symmv_sort	12	37	3
gsl_eigen_symmv_alloc	7	21	1
gsl_eigen_symmv_free	1	7	1
gsl_eigen_symmv	117	52	4
gsl_linalg_symmtd_decomp	62	26	2
gsl_linalg_symmtd_unpack	21	28	5
gsl_blas_dsyr2	4	9	5
gsl_blas_daxpy	3	6	3
gsl_blas_ddot	3	6	3
gsl_linalg_householder_transform	12	22	1
gsl_blas_dnrm2	2	2	1
cblas_dnrm2	1	4	3
gsl_blas_dscal	2	2	2
cblas_dscal	1	4	4
gsl_linalg_householder_hm	12	26	3
chop_small_elements	3	9	3
qrstep	18	63	5
gsl_blas_dsymv	4	9	6
cblas_dsymv	1	4	12
cblas_dsyr2	1	4	10
cblas_ddot	1	8	5
create_givens	3	14	4
cblas_daxpy	1	4	6

Табела А.5 – Метрики за апликацијата А1

модул	излезни параметри	глобални променливи	повикува модули
double * make_array_ptsxy	1	0	0
double * make_array_fbrtxy	1	0	0
double** make_matrix_Txy	1	0	0
struct return_objects thcheby	6	0	3
print_n_elements_from_array	0	0	0
double** transpose_matrix	1	0	0
double** diagonal_matrix	1	0	0
double** matrix_multiplication	1	0	0
double* divide_vector_number	1	0	0
double **multiple_element_matrix	1	0	0
double **identity_matrix	1	0	0
double ***array_matrices_multiply	1	0	2
double **matrix_addition	1	0	0
double ***array_matrices_add	1	0	2
struct file_plus_rows return_data_file	2	0	0
void swap_el	0	0	0
int compare_file_row	1	0	0
struct file_plus_rows sortarray	2	0	3
double *get_eigenvalues	1	0	4
double f	1	1	0
double best_nearby	1	0	1
int hooke	1	0	1
double ** interpolation	1	0	0
double * prepare_PES_array	1	1	5
void DVR2Dv1	0	0	9
gsl_eigen_symmv_sort	1	0	3
gsl_eigen_symmv_alloc	1	0	0
gsl_eigen_symmv_free	0	0	0
gsl_eigen_symmv	1	0	4
gsl_linalg_symmtd_decomp	1	0	5
gsl_linalg_symmtd_unpack	1	0	1
gsl_blas_dsyr2	1	0	1
gsl_blas_daxpy	1	0	1
gsl_blas_ddot	1	0	1
gsl_linalg_householder_transform	1	0	2
gsl_blas_dnrm2	1	0	1
cblas_dnrm2	1	0	0
gsl_blas_dscal	0	0	1
cblas_dscal	0	0	0
gsl_linalg_householder_hm	1	0	2
chop_small_elements	0	0	0
qrstep	0	0	1
gsl_blas_dsymv	1	0	1
cblas_dsymv	0	0	0
cblas_dsyr2	0	0	0
cblas_ddot	1	0	0
create_givens	0	0	0
cblas_daxpy	0	0	0

Табела А.6 – Метрики за апликацијата А1

модул	повикан во модули	зависност од други модули
double * make_array_ptsxy	1	0.833333333
double * make_array_fbrtxy	1	0.75
double** make_matrix_Txy	1	0.75
struct return_objects thcheby	1	0.9375
print_n_elements_from_array	1	0.75
double** transpose_matrix	1	0.75
double** diagonal_matrix	2	0.8
double** matrix_multiplication	1	0.8
double* divide_vector_number	1	0.8
double **multiple_element_matrix	1	0.8
double **identity_matrix	1	0.666666667
double ***array_matrices_multiply	1	0.833333333
double **matrix_addition	1	0.8
double ***array_matrices_add	1	0.75
struct file_plus_rows return_data_file	1	0.75
void swap_el	1	0.8
int compare_file_row	1	0.875
struct file_plus_rows sortarray	2	0.909090909
double *get_eigenvalues	1	0.875
double f	2	0.875
double best_nearby	1	0.888888889
int hooke	1	0.909090909
double ** interpolation	1	0.833333333
double * prepare_PES_array	1	0.875
void DVR2Dv1	1	0.941176471
gsl_eigen_symmv_sort	1	0.875
gsl_eigen_symmv_alloc	1	0.666666667
gsl_eigen_symmv_free	1	0.5
gsl_eigen_symmv	1	0.9
gsl_linalg_symmtd_decomp	1	0.888888889
gsl_linalg_symmtd_unpack	1	0.875
gsl_blas_dsyr2	1	0.875
gsl_blas_daxpy	2	0.857142857
gsl_blas_ddot	2	0.857142857
gsl_linalg_householder_transform	1	0.8
gsl_blas_dnrm2	1	0.75
cblas_dnrm2	1	0.8
gsl_blas_dscal	1	0.75
cblas_dscal	1	0.8
gsl_linalg_householder_hm	1	0.857142857
chop_small_elements	1	0.75
qrstep	1	0.857142857
gsl_blas_dsymv	1	0.888888889
cblas_dsymv	1	0.923076923
cblas_dsyr2	1	0.909090909
cblas_ddot	1	0.857142857
create_givens	1	0.8
cblas_daxpy	1	0.857142857

Библиографија

[1] Halstead metrics tool. URL https://sourceforge.net/projects/halsteadmetricstool/

[2] IEEE Guide to Software Requirements Specifications. IEEE Std 8301984 pp. 0_1– (1984). DOI 10.1109/IEEESTD.1984.119205

[3] IEEE Standard Glossary of Software Engineering Terminology. The Institute of Electrical and Electronics Ehgineers, New York, NY, USA (1990). [Online]. Available: http://www.idi.ntnu.no/grupper/su/publ/ese/ieee-se-glossary-610.12-1990.pdf (current November 2012)

[4] IEEE Standard Glossary of Software Engineering Terminology. Tech. rep. (1990). DOI 10.1109/ieeestd.1990.101064. URL http://dx.doi.org/10.1109/ieeestd.1990.101064

[5] IEEE Recommended Practice for Software Requirements Specifications. IEEE Std 8301998 pp. 1–40 (1998). DOI 10.1109/IEEESTD.1998.88286

[6] IEEE Draft Standard for Systems and Software Engineering – Requirements for Acquirers and Suppliers of User Documentation. IEEE P26512/D3, November 2010 pp. 1–55 (2010)

[7] CVS - concurrent versions system. http://www.nongnu.org/cvs/ (2016). Last accessed 23 July 2016

[8] Git. https://git-scm.com/ (2016). Last accessed 23 July 2016

[9] Subversion. https://subversion.apache.org/ (2016). Last accessed 23 July 2016

[10] Assoc for Info and Image Mgmt Intl : The Global Community of Information Professionals (2013). URL http://www.aiim.org/community/wiki/view/Index-W

[11] 12651-2: ISO/DIS 12651-2, Electronic imaging – Vocabulary – Part 2: Document workflow (2013). URL http://www.iso.org/iso/catalogue_detail.htm?csnumber=42673

[12] Agarwal, B., Tayal, S., Gupta, M.: Software engineering and testing. Jones & Bartlett Learning (2010)

[13] Alexeev, Y., Allan, B.A., Armstrong, R.C., Bernholdt, D.E., Dahlgren, T.L., Gannon, D., Janssen, C.L., Kenny, J.P., Krishnan, M., Kohl, J.A., Kumfert, G., McInnes, L.C., Nieplocha, J., Parker, S.G., Rasmussen, C., Windus, T.L.: Component-based software for high-performance scientific computing. Journal of Physics: Conference Series 16, 536—-540 (2005)

[14] Almas, Q.L., Keefe, B.L., Profitt, T., Pearson, J.K.: Choosing an appropriate model chemistry in a big data context: Application to dative bonding. Computational and Theoretical Chemistry 1085, 46–55 (2016)

[15] Ammann, P., Offutt, J.: Introduction to software testing. Cambridge University Press (2016)

[16] Andersen, P.B., Prange, F., Serritzlew, S.: Software engineering as a part of scientific practice. University of Aarhus, Denmark (2011). [Online]. Available: http://imv.au.dk/ pba/Homepagematerial/publicationfolder/softwareengineering.pdf (current November 2012)

[17] Apache Software Foundation: Hadoop. URL https://hadoop.apache.org

[18] Arora, A.: Agile automation testing (2008)

[19] Artho, C., Barringer, H., Goldberg, A., Havelund, K., Khurshid, S., Lowry, M., Pasareanu, C., Rosu, G., Sen, K., Visser, W., Washington, R.: Combining test case generation and runtime verification. Theor. Comput. Sci. 336(2-3), 209–234 (2005). DOI 10.1016/j.tcs.2004.11.007. URL http://dx.doi.org/10.1016/j.tcs.2004.11.007

[20] Asadzadeh, P., Buyya, R., Kei, C.L., Nayar, D., Venugopal, S.: Global grids and software toolkits: A study of four grid middleware technologies. Wiley Press, New Jersey, USA (2005)

[21] Avrunin, G.S., Siegel, S.F., Siegel, A.R.: Finite-state verification for high performance computing. In: Proceedings of the second international workshop on Software engineering for high performance computing system applications, pp. 68–72. ACM, New York, NY, USA (2005)

[22] Bader, R.F.W., Keith, T.A.: Properties of atoms in molecules: Magnetic susceptibilities. The Journal of Chemical Physics 99(5), 3683–3693 (1993). DOI http://dx.doi.org/10.1063/1.466166

[23] Baker, M., Buyya, R., Laforenza, D.: Grids and grid technologies for wide-area distributed computing. Software: Practice and Experience 32(15), 1437–1466 (2002)

[24] Barker, A., Van Hemert, J.: Scientific workflow: a survey and research directions. In: Proceedings of the 7th international conference on Parallel processing and applied mathematics, PPAM'07, pp. 746–753. Springer-Verlag, Berlin, Heidelberg (2008). URL http://dl.acm.org/citation.cfm?id=1786194.1786281

[25] Basili, V.R., Carver, J.C., Cruzes, D., Hochstein, L.M., Hollingsworth, J.K., Shull, F., Zelkowitz, M.V.: Understanding the high performance computing community: A software engineer's perspective. IEEE Software 25(4), 29–36 (2008)

[26] Bastian, P., Blatt, M., Dedner, A., Engwer, C., Klöfkorn, R., Ohlberger, M., Sander, O.: A generic grid interface for parallel and adaptive scientific computing. part i: abstract framework. Computing 82(2-3), 103–119 (2008)

[27] Baxter, R.: Software engineering is software engineering. In: Proceedings of the First International Workshop on Software Engineering for High Performance Computing System Application, pp. 14–18. IEE, Edinburgh, Scotland, United Kingdom (2004)

[28] Berendsen, H.J.C., Postma, J.P.M., van Gunsteren, W.F., Hermans, J.: Intermolecular forces. B. Pullman ed., Reidel, Dordrecht (1981)

[29] Bharathi, S., Chervenak, A., Deelman, E., Mehta, G., Su, M.H., Vahi, K.: Characterization of scientific workflows. In: Workflows in Support of Large-Scale Science, 2008. WORKS 2008. Third Workshop on, pp. 1–10 (2008). DOI 10.1109/WORKS.2008.4723958

[30] Bird, C., Nagappan, N., Devanbu, P., Gall, H., Murphy, B.: Does distributed development affect software quality?: an empirical case study of windows vista. Communications of the ACM 52(8), 85–93 (2009)

[31] Biswas, S., Mall, R., Satpathy, M., Sukumaran, S.: Regression test selection techniques: A survey. Informatica 35(3) (2011)

[32] Bittner, E.: Quantum dynamics: applications in biological and materials systems. CRC Press (2009). URL http://books.google.mk/books?id=wKrvAAAAMAAJ

[33] Bittner, E.: Quantumdvr.nb - A series of demonstrations and utility routines for computing eigenvalues and performing wave-packet propagation in an two-dimensional potential well using the Discrete Variable Representation. (2009). URL http://k2.chem.uh.edu/quantum_dynamics/Downloads.html

[34] Boisvert, R.F., Tang, P.T.P.: The Architecture of Scientific Software: IFIP TC2/WG2. 5 Working Conference on the Architecture of Scientific Software October 2–4, 2000, Ottawa, Canada, vol. 60. Springer (2013)

[35] Booth, S., Henty, D.: Verfication strategies for high performance computing software. In: Proceedings of the First International Workshop on Software Engineering for High Performance Computing System Applications, pp. 24–26. IEE, Edinburgh, Scotland, United Kingdom (2004)

[36] Bote-Lorenzo, M.L., Dimitriadis, Y.A., Gómez-Sánchez, E.: Grid characteristics and uses: a grid definition. In: Grid Computing, pp. 291–298. Springer (2004)

[37] Britannica, E.: Encyclopaedia Britannica. Encyclopædia Britannica (1973)

[38] Buck, J.B., Watkins, N., LeFevre, J., Ioannidou, K., Maltzahn, C., Polyzotis, N., Brandt, S.: Scihadoop: Array-based query processing in hadoop. In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, SC '11, pp. 66:1–66:11. ACM, New York, NY, USA (2011). DOI 10.1145/2063384.2063473. URL http://doi.acm.org/10.1145/2063384.2063473

[39] Bunch, C., Drawert, B., Norman, M.: Mapscale: a cloud environment for scientific computing. University of California, Computer Science Department, Tech. Rep (2009)

[40] Cau, A., Moszkowski, B., Zedan, H.: Interval temporal logic (2002). URL http://citeseerx.ist.psu.edu/viewdoc/download?rep=rep1&type=pdf&doi=10.1.1.4.6669

[41] Chen, T.Y., Cheung, S.C., Yiu, S.M.: Metamorphic testing: a new approach for generating next test cases. Tech. rep., Technical Report HKUST-CS98-01, Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong (1998)

[42] Chin, L.S., Worth, D.J., Greenough, C.: A survey of software testing tools for computational science. RAL Technical Reports (2007). URL [Online].Available:http://www.softeng.rl.ac.uk/media/uploads/publications/2010/06/swtesting.pdf(currentNovember2012)

[43] Chittimalli, P.K., Harrold, M.J.: Regression test selection on system requirements. In: Proceedings of the 1st India software engineering conference, pp. 87–96. ACM (2008)

[44] Christensen, H.B.: Flexible, reliable software: using patterns and agile development. CRC Press (2011)

[45] Christopherson, L., Idaszak, R., Ahalt, S.: Developing Scientific Software through the Open Community Engagement Process (2013). URL http://dx.doi.org/10.6084/m9.figshare.790723

[46] Clarke, E.M., Wing, J.M.: Formal methods: State of the art and future directions. ACM Comput. Surv. 28(4), 626–643 (1996). DOI 10.1145/242223.242257. URL http://doi.acm.org/10.1145/242223.242257

[47] Coleman, D., Ash, D., Lowther, B., Oman, P.: Using metrics to evaluate software system maintainability. Computer 27(8), 44–49 (1994)

[48] Collet, P., Křikava, F., Montagnat, J., Blay-Fornarino, M., Manset, D.: Issues and scenarios for self-managing grid middleware. In: Proceedings of the 2nd workshop on Grids meets autonomic computing, pp. 1–10. ACM (2010)

[49] Committee, I.C.S.S.E.S., Board, I.S.S.: Ieee recommended practice for software requirements specifications. Institute of Electrical and Electronics Engineers (1998)

[50] Committee, I.C.S.S.E.T.: IEEE standard glossary of software engineering terminology. ANSI/IEEE. Institute of Electrical and Electronics Engineers (1983). URL https://books.google.mk/books?id=CM1WAAAAMAAJ

[51] bugzilla.org contributors, I.: Bugzilla. URL https://www.bugzilla.org/

[52] Coutinho, K., Canuto, S.: DICE: A Monte Carlo program for molecular liquid simulation. University of São Paulo, Brazil (1997)

[53] Coutinho, K., Georg, H., Fonseca, T., Ludwig, V., Canuto, S.: An efficient statistically converged average configuration for solvent effects. Chemical Physics Letters 437(1–3), 148 – 152 (2007). DOI http://dx.doi.org/10.1016/j.cplett.2007.02.012

[54] Curcin, V., Ghanem, M.: Scientific workflow systems-can one size fit all? In: 2008 Cairo International Biomedical Engineering Conference, pp. 1–9. IEEE (2008)

[55] Dabrowski, C.: Reliability in grid computing systems. Concurrency and Computation: Practice and Experience 21(8), 927–959 (2009)

[56] De Montfort University: Anatempura tool for runtime verification and animation (2004). URL http://www.tech.dmu.ac.uk/STRL/research/software/anatempura.pdf

[57] Dean, J., Ghemawat, S.: Mapreduce: Simplified data processing on large clusters. Commun. ACM 51(1), 107–113 (2008). DOI 10.1145/1327452.1327492. URL http://doi.acm.org/10.1145/1327452.1327492

[58] Deelman, E., Gannon, D., Shields, M., Taylor, I.: Workflows and e-science: An overview of workflow system features and capabilities. Future Gener. Comput. Syst. 25(5), 528–540 (2009). DOI 10.1016/j.future.2008.06.012. URL http://dx.doi.org/10.1016/j.future.2008.06.012

[59] Deshpande, A., Anand, N., Manisha, V., Garje, G.: Improving software quality with agile testing. Emergence 1(22) (2010)

[60] DeSouza, J., Kuhn, B., Supinsk, B.: Automated, scalable debugging of mpi programs with intel® message checker. In: Proceedings of the second international workshop on Software engineering for high performance computing system applications, pp. 78–82. ACM, New York, NY, USA (2005)

[61] Ditchfield, R.: Self-consistent perturbation theory of diamagnetism. Molecular Physics 27(4), 789–807 (1974). DOI 10.1080/00268977400100711

[62] Dykstra, C.: Quantum chemistry and molecular spectroscopy. Prentice Hall PTR (1992)

[63] Ekanayake, J., Pallickara, S., Fox, G.: Mapreduce for data intensive scientific analyses. In: Proceedings of the 2008 Fourth IEEE International Conference on eScience, ESCIENCE '08, pp. 277–284. IEEE Computer Society, Washington, DC, USA (2008). DOI 10.1109/eScience.2008.59. URL http://dx.doi.org/10.1109/eScience.2008.59

[64] El-kustaban, A., Moszkowski, B., Cau, A.: Specification analysis of transactional memory using itl and anatempura. Lecture Notes in Engineering and Computer Science 2195(1), 176–181 (2012)

[65] Elmaallam, M.: Towards a model of maturity for is risk management. International Journal of Computer Science & Information Technology 3(4) (2011)

[66] Fei, X., Lu, S.: A dataflow-based scientific workflow composition framework. IEEE Trans. Serv. Comput. 5(1), 45–58 (2012). DOI 10.1109/TSC.2010.58. URL http://dx.doi.org/10.1109/TSC.2010.58

[67] Forum, M.P.: Mpi: A message-passing interface standard. Tech. rep., Knoxville, TN, USA (1994)

[68] Foster, I., Kesselman, C.: The globus toolkit. The grid: blueprint for a new computing infrastructure pp. 259–278 (1999)

[69] Foundation, T.A.S.: Apache™ hadoop. URL https://hadoop.apache.org/docs/r2.6.1/api/org/apache/hadoop/io/WritableComparable.html

[70] Fowler, M., Highsmith, J.: The agile manifesto. Software Development 9(8), 28–35 (2001)

[71] Free Software Foundation: GNU Affero General Public License (2016). URL http://www.gnu.org/licenses/agpl.html

[72] Gerlec, Č., Rakić, G., Budimac, Z., Heričko, M.: A programming language independent framework for metrics-based software evolution and analysis. Computer Science and Information Systems 9(3), 1155–1186 (2012)

[73] Ghosh, S., Sharma, H., Mohabay, V.: Analysis and modeling of change management process model. International Journal of Software Engineering and Its Applications 2 (2011)

[74] Goble, C.A., De Roure, D.C.: myexperiment: social networking for workflow-using e-scientists. In: Proceedings of the 2nd workshop on Workflows in support of large-scale science, WORKS '07, pp. 1–2. ACM, New York, NY, USA (2007). DOI 10.1145/1273360.1273361. URL http://doi.acm.org/10.1145/1273360.1273361

[75] Gorlatch, S., Dünnweber, J.: From grid middleware to grid applications: Bridging the gap with hocs. In: Future Generation Grids, pp. 241–261. Springer (2006)

[76] Groote, J., Osaiweran, A., Wesselius, J.: Benefits of applying formal methods to industrial control software. Computer Science Report 11(04) (2011)

[77] Habela, P., Subieta, K.: Oodbms metamodel supporting configuration management of large applications. In: International Conference on Object-Oriented Information Systems, pp. 40–52. Springer (2002)

[78] Hall, A.: Realising the benefits of formal methods. In: K.K. Lau, R. Banach (eds.) Formal Methods and Software Engineering, Lecture Notes in Computer Science, vol. 3785, pp. 1–4. Springer Berlin Heidelberg (2005). DOI 10.1007/11576280_1. URL http://dx.doi.org/10.1007/11576280_1

[79] Halstead, M.H.: Elements of software science, vol. 7. Elsevier New York (1977)

[80] Hannay, J.E., MacLeod, C., Singer, J., Langtangen, H., Pfahl, D., Wilson, G.: How do scientists develop and use scientific software? In: Proceedings of the 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering, pp. 1–8. IEEE Computer Society, Washington, DC, USA (2009)

[81] Harish, R., Madhu, B., Lokesha, V.: A sophisticated study on best practices of agile software testing. International Journal of Electronics communication and computer engineering 3(1), 30 (2012)

[82] He, C.: Molecular dynamics simulation based on hadoop mapreduce (2011)

[83] Heaton, D., Carver, J.C., Bartlett, R., Oakes, K., Hochstein, L.: The relationship between development problems and use of software engineering practices in computational science & engineering: A survey. Tech. Rep. SERG-2012-05, Department of Computer Science, The University of Alabama (2012). URL http://software.eng.ua.edu/reports/SERG-2012-05

[84] Heaton, D., Carver, J.C., Bartlett, R., Oakes, K., Hochstein, L.: The relationship between development problems and use of software engineering practices in computational science & engineering: A survey. In: Proceedings of the E-Science, IEEE 8th International Conference (2012)

[85] Heimdahl, E.V.W.M.P., Saad, Y.: From models to efficient code: It's all in the middle. In: First International Workshop On Software Engineering for High Performance Computing System Applications, p. 54 (2004)

[86] Heitlager, I., Kuipers, T., Visser, J.: A practical model for measuring maintainability. In: Quality of Information and Communications Technology, 2007. QUATIC 2007. 6th International Conference on the, pp. 30–39. IEEE (2007)

[87] Hernández, F., Bangalore, P., Reilly, K.: Automating the development of scientific applications using domain-specific modeling. In: Proceedings of the second international workshop on Software engineering for high performance computing system applications, pp. 50–54. ACM, New York, NY, USA (2005)

[88] Hochstein, L., Basili, V.: The asc-alliance projects: A case study of large-scale parallel scientific code development. Computer 41(3), 50—-58 (2008)

[89] Hoffmann, V., Lichter, H., Nyen, A.: Processes and practices for quality scientific software projects. In: Proceedings of 3rd International Workshop on Academic Software Development Tools WASDeTT-3, Antwerp, pp95-108 (2009)

[90] Hook, D., Kelly, D.: Testing for trustworthiness in scientific software. In: Proceedings of the 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering, pp. 59–64. IEEE Computer Society (2009)

[91] Howison, J., Herbsleb, J.: Socio-technical logics of correctness in the scientific software development ecosystem (2010)

[92] Howison, J., Herbsleb, J.: Scientific software production: incentives and collaboration. In: Proceedings of the 2011 ACM conference on Computer supported cooperative work, pp. 513–522. ACM, New York, NY, USA (2011)

[93] Huang, D., Lapp, H.: Software Engineering as Instrumentation for the Long Tail of Scientific Software. ArXiv e-prints (2013)

[94] Iee, E.: IEEE Std 1061-1998. IEEE Standard for a Software Quality Metrics Methodology (1998)

[95] InfoEther: Pmd duplicate code detector. URL https://pmd.github.io/pmd-5.4.1/usage/cpd-usage.html

[96] Iosup, A., Epema, D.: Build-and-test workloads for grid middleware: Problem, analysis, and applications. In: Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid'07), pp. 205–213. IEEE (2007)

[97] ISO/IEC: ISO/IEC 9126. Software engineering – Product quality. ISO/IEC (2001)

[98] Issarny, V., Caporuscio, M., Georgantas, N.: A perspective on the future of middleware-based software engineering. In: 2007 Future of Software Engineering, pp. 244–258. IEEE Computer Society (2007)

[99] Jaakkola, H., Thalheim, B.: Framework for high-quality software design and development: a systematic approach. IET software 4(2), 105–118 (2010)

[100] Jaeger, E.: Study of the Benefits of Using Deductive Formal Methods for Secure Developments (2010). URL http://books.google.mk/books?id=EU2KMwEACAAJ

[101] Jalis, A.: Cutest: C unit testing framework 1.5 (2013) (2016)

[102] Jalote, P., Palit, A., Kurien, P.: Timeboxing: A process model for iterative software development. Journal of Systems and Software 2004, 3 (2004)

[103] Jamil, H., Islam, A.: The power of declarative languages: a comparative exposition of scientific workflow design using bioflow and taverna. In: 2009 Congress on Services-I, pp. 322–329. IEEE (2009)

[104] Johnson, C., Patel, R., Radcliffe, D., Lee, P., Nguyen, J.: Establishing qualitative software metrics in department of the navy programs. Tech. rep., DTIC Document (2015)

[105] Jones, C.: Programming languages table, release 8.2. Software Productivity Research, Burlington, MA (1996)

[106] Jureczko, M.: The level of agility in testing process in a large scale financial software project. Software engineering techniques in progress, Oficyna Wydawnicza Politechniki Wroc pp. 139–152 (2008)

[107] Kanewala, U., Bieman, J.M.: Testing scientific software: A systematic literature review. Information and software technology 56(10), 1219–1232 (2014)

[108] Karloff, H., Suri, S., Vassilvitskii, S.: A model of computation for mapreduce. In: Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '10, pp. 938–948. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (2010). URL http://dl.acm.org/citation.cfm?id=1873601.1873677

[109] Kaur, G., Bahl, K.: Software reliability, metrics, reliability improvement using agile process. International Journal of Innovative Science, Engineering and Technology 1(3), 143–7 (2014)

[110] Keith, T., Bader, R.: Calculation of magnetic response properties using atoms in molecules. Chemical Physics Letters 194(1–2), 1 – 8 (1992). DOI http://dx.doi.org/10.1016/0009-2614(92)85733-Q

[111] Keith, T.A., Bader, R.F.: Calculation of magnetic response properties using a continuous set of gauge transformations. Chemical Physics Letters 210(1–3), 223 – 231 (1993). DOI http://dx.doi.org/10.1016/0009-2614(93)89127-4

[112] Kelly, D., Sanders, R.: Assessing the quality of scientific software. In: First International Workshop on Software Engineering for Computational Science and Engineering (2008)

[113] Khan, A.A., Keung, J.: Systematic review of success factors and barriers for software process improvement in global software development. IET Software (2016)

[114] Kiel, L., Eng, P.: Experiences in distributed development: a case study. In: International Workshop on Global Software Development at ICSE, pp. 44–47. Citeseer (2003)

[115] Kocevski, V., Pejov, L.: Anharmonic vibrational frequency shifts upon interaction of phenol(+) with the open shell ligand o2. the performance of dft methods versus mp2. The Journal of Physical Chemistry A 116(8), 1939–1949 (2012). DOI 10.1021/jp209801s. URL http://pubs.acs.org/doi/abs/10.1021/jp209801s

[116] Kohls, E., Mishev, A., Pejov, L.: Solvation of fluoroform and fluoroform–dimethylether dimer in liquid krypton: A theoretical cryospectroscopic study. The Journal of Chemical Physics 139(5), 054504 (2013). DOI http://dx.doi.org/10.1063/1.4816282. URL http://scitation.aip.org/content/aip/journal/jcp/139/5/10.1063/1.4816282

[117] Kommeren, R., Parviainen, P.: Philips experiences in global distributed software development. Empirical Software Engineering 12(6), 647–660 (2007)

[118] Koteska, B., Jakimovski, B., Mishev, A.: Building Scientific Workflows on the Grid: A Comparison between OpenMole and Taverna. In: RO-LCG 2014 Conference (2014)

[119] Koteska, B., Misev, A.: Change Management and Version Control of Scientific Applications. International Journal of Computer Science & Information Technology - IJCSIT 6(2), 153–161 (2014). DOI 10.5121/ijcsit.2014.6211

[120] Koteska, B., Mishev, A.: Agile Software Testing Technologies in a Large Scale Project. In: Local Proceedings of the Fifth Balkan Conference in Informatics, BCI 2012, vol. 920, pp. 121–124. Faculty of Sciences, University of Novi Sad, Serbia (2012)

[121] Koteska, B., Mishev, A.: A Software Engineering Perspective for Higher Quality Grid Distributed Development. In: Proceedings of the 10th Conference for Informatics and Information Technology, pp. 247–251. Faculty of Computer Science and Engineering, Skopje, Macedonia (2013)

[122] Koteska, B., Mishev, A.: Software engineering practices and principles to increase quality of scientific applications. In: S. Markovski, M. Gusev (eds.) ICT Innovations 2012, Advances in Intelligent Systems and Computing, vol. 207, pp. 245–254. Springer Berlin Heidelberg (2013). DOI 10.1007/978-3-642-37169-1_24. URL http://dx.doi.org/10.1007/978-3-642-37169-1_24

[123] Koteska, B., Mishev, A., Pejov, L.: A Robust Hybrid Statistical Physics-Quantum Chemical Approach to Magnetic Properties of Complex Aqueous Aluminimum (III) Species: Implementation on HPC Environment. In: Women in HPC, ISC High Performance (2015)

[124] Koteska, B., Mishev, A., Pejov, L.: Framework for Developing Scientific Applications: Solving 1D and 2D Schrödinger Equation by using Discrete Variable Representation Method. In: Proceedings of the First International Conference on Advances and Trends in Software Engineering - SOFTENG, pp. 93–99 (2015)

[125] Koteska, B., Mishev, A., Pejov, L.: Magnetic Response Properties of Aqueous Aluminum(III) Ion: A Hybrid Statistical Physics Quantum Mechanical Approach Implementing the Map-Reduce Computational Technique. In: A.M. Bogdanova, D. Gjorgjevikj (eds.) ICT Innovations 2014: World of Data, pp. 33–43. Springer International Publishing, Cham (2015). DOI 10.1007/978-3-319-09879-1_4. URL https://doi.org/10.1007/978-3-319-09879-1_4

[126] Koteska, B., Pejov, L., Mishev, A.: Software Engineering Solutions for Improving the Regression Testing Methods in Scientific Applications Development. In: Proceedings of the 2nd Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications, SQAMIA 2013, vol. 1053, pp. 53–61. Faculty of Sciences, University of Novi Sad, Serbia (2013)

[127] Koteska, B., Pejov, L., Mishev, A.: Average Vibrational Potentials of Oscillators in Condensed-matter Environments using Hadoop. In: Proceedings of the 11th Conference for Informatics and Information Technology, pp. 311–314. Faculty of Computer Science and Engineering, Skopje, Macedonia (2014)

[128] Koteska, B., Pejov, L., Mishev, A.: Formal Specification of Scientific Applications Using Interval Temporal Logic. In: Proceedings of the 3rd Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications, SQAMIA 2014, vol. 1266, pp. 29–37. Faculty of Sciences, University of Novi Sad, Serbia (2014)

[129] Koteska, B., Pejov, L., Mishev, A.: Scientific Software Testing: A Practical Example. In: Proceedings of the 4th Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications, SQAMIA 2015, vol. 1375, pp. 27–34. Faculty of Sciences, University of Novi Sad, Serbia (2015)

[130] Koteska, B., Pejov, L., Mishev, A.: Quantitative measurement of scientific software quality: Definition of a novel quality model. International Journal of Software Engineering and Knowledge Engineering pp. accepted, 2016 JCR Impact Factor 0.299 (2018)

[131] Krajnc, A., Heričko, M., Gerlec, Č., Goljat, U., Polančič, G.: Experimental investigation of the quality and productivity of software factories based development. Computer Science and Information Systems 9(2), 667–689 (2012)

[132] Krishnan, S., Stearn, B., Bhatia, K., Baldridge, K.K., Li, W.W., Arzberger, P.: Opal: Simpleweb services wrappers for scientific applications. In: Web Services, 2006. ICWS'06. International Conference on, pp. 823–832. IEEE (2006)

[133] Kumar, R., Gattaiah, D.Y., Shahi, S., Nagendra, T.A.: Improving software quality assurance using bug tracking system. International Journal of Computer Science and Information Technologies 4(3), 492–497 (2013)

[134] Kyriakidou, A., Venters, W., et al.: Distributed large-scale systems development: Exploring the collaborative development of the particle physics grid. In: 5th International conference on e-Social Science, Cologne. Citeseer (2009)

[135] Kyriakidou-Zacharoudiou, A.: Distributed development of large-scale distributed systems: the case of the particle physics grid. Ph.D. thesis, London School of Economics (2011)

[136] Lanubile, F.: Collaboration in distributed software development. In: Software Engineering, pp. 174–193. Springer (2009)

[137] von Laszewski, G., Amin, K.: Grid middleware. Middleware for Communications p. 109 (2004)

[138] Laure, E., Hemmer, F., Prelz, F., Beco, S., Fisher, S., Livny, M., Guy, L., Barroso, M., Buncic, P., Kunszt, P., et al.: Middleware for the next generation

grid infrastructure. Computing in High Energy Physics and Nuclear Physics (CHEP 2004) (2004)

[139] Letondal, C., Zdun, U.: Anticipating scientific software evolution as a combined technological and design approach. In: Second International Workshop on Unanticipated Software Evolution (2003)

[140] Li, Y., Narayan, N., Helming, J., Koegel, M.: A domain specific requirements model for scientific computing: Nier track. In: Software Engineering (ICSE), 2011 33rd International Conference on, pp. 848–851 (2011). DOI 10.1145/1985793.1985922

[141] Light, J.C., Carrington Jr, T.: Discrete-variable representations and their utilization. Advances in Chemical Physics 114, 263–310 (2000)

[142] Luksch, P., Maier, U., Rathmayer, S., Weidmann, M.: Software engineering methods for parallel and distributed scientific computing. In: Proceedings of the International Conference and Exhibition on High-Performance Computing and Networking, HPCN Europe 1996, pp. 508–513. Springer-Verlag, London, UK, UK (1996). URL http://dl.acm.org/citation.cfm?id=645560.658551

[143] Luksch, P., Maier, U., Rathmayer, S., Weidmann, M., Unger, F.: Software engineering methods for parallel applications in scientific computing project. Project Report, Shaker-Verlag, Aachen (1988). [Online]. Available: http://wwwbode.informatik.tu-muenchen.de/archiv/artikel/pdse96/PDSE96.ps.gz (current November 2012)

[144] Lämmel, R.: Google's mapreduce programming model — revisited. Science of Computer Programming 70(1), 1 – 30 (2008). DOI http://dx.doi.org/10.1016/j.scico.2007.07.001. URL http://www.sciencedirect.com/science/article/pii/S0167642307001281

[145] Mäkäräinen, M.: Software change management process in the development of embedded software. VTT PUBLICATIONS 4(1), 6 (2000)

[146] Marques, O., Drummond, T.: Building a software infrastructure for computational science applications: lessons and solutions. In: Proceedings of the second international workshop on Software engineering for high performance computing system applications, pp. 40–44. ACM, New York, NY, USA (2005)

[147] McCabe, T.J.: A complexity measure. IEEE Transactions on software Engineering (4), 308–320 (1976)

[148] Miriyala, K., Harandi, M.: Automatic derivation of formal software specifications from informal descriptions. IEEE Transactions on Software Engineering 17(10), 1126–1142 (1991). DOI http://doi.ieeecomputersociety.org/10.1109/32.99198

[149] Moszkowski, B.: Executing temporal logic programs. In: S. Brookes, A. Roscoe, G. Winskel (eds.) Seminar on Concurrency, Lecture Notes in Computer Science, vol. 197, pp. 111–130. Springer Berlin Heidelberg (1985). DOI 10.1007/3-540-15670-4_6. URL http://dx.doi.org/10.1007/3-540-15670-4_6

[150] Moszkowski, B., Manna, Z.: Reasoning in interval temporal logic. In: E. Clarke, D. Kozen (eds.) Logics of Programs, Lecture Notes in Computer Science, vol. 164, pp. 371–382. Springer Berlin Heidelberg (1984). DOI 10.1007/3-540-12896-4_374. URL http://dx.doi.org/10.1007/3-540-12896-4_374

[151] Munassar, N.M.A., Govardhan, A.: A comparison between five models of software engineering. IJCSI 5, 95–101 (2010)

[152] Nawaz, A., Malik, K.M.: Software testing process in agile development. Computer Science Master Thesis, Blekinge Institute of Technology (2008)

[153] Neely, R.: Practical software quality engineering on a large multidisciplinary hpc development team. In: Proceedings of the First International Workshop On Software Engineering for High Performance Computing System Applications, pp. 19–23. IEE, Edinburgh, Scotland, United Kingdom (2004)

[154] Oberkampf, W.L., Trucano, T.G., Hirsch, C.: Verification, validation, and predictive capability in computational engineering and physics. Applied Mechanics Reviews 57(5), 345–384 (2004)

[155] Oinn, T., Greenwood, M., Addis, M., Alpdemir, M.N., Ferris, J., Glover, K., Goble, C., Goderis, A., Hull, D., Marvin, D., Li, P., Lord, P., Pocock, M.R., Senger, M., Stevens, R., Wipat, A., Wroe, C.: Taverna: lessons in creating a workflow environment for the life sciences: Research articles. Concurr. Comput. : Pract. Exper. 18(10), 1067–1100 (2006). DOI 10.1002/cpe.v18:10. URL http://dx.doi.org/10.1002/cpe.v18:10

[156] Oman, P., Hagemeister, J.: Metrics for assessing a software system's maintainability. In: Software Maintenance, 1992. Proceerdings., Conference on, pp. 337–344. IEEE (1992)

[157] Ostroff, J.S., Paige, R.F.: Formal methods in the classroom: The logic of real-time software design. In: Proceedings of the Third IEEE Real-Time Systems Education Workshop, RTEW '98, pp. 63–. IEEE Computer Society, Washington, DC, USA (1998). URL http://dl.acm.org/citation.cfm?id=554225.828878

[158] Out, S.U.S.: Mapreduce programming model. (2010)

[159] O'Regan, G.: Software engineering institute (sei). In: Pillars of Computing, pp. 195–205. Springer (2015)

[160] Parashar, M., Browne, J.C.: Systems engineering for high performance computing software: The hdda/dagh infrastructure for implementation of parallel

structured adaptive mesh refinement. In: Structured Adaptive Mesh Refinement Grid Methods, IMA Volumes in Mathematics and its Applications, pp. 1–18. Springer-Verlag New York Inc., New York, NY, USA (1997)

[161] PCMag: Definition of commercial application (2016). URL http://www.pcmag.com/encyclopedia/term/40059/commercial-software

[162] Phadke, A.A., Allen, E.B.: Predicting risky modules in open-source software for high-performance computing. In: Proceedings of the second international workshop on Software engineering for high performance computing system applications, pp. 60–64. ACM, New York, NY, USA (2005)

[163] van der Poll, J.A.: Formal methods in software development: a road less travelled. South African Computer Journal 45, 40–52 (2010). URL http://dblp.uni-trier.de/db/journals/saj/saj45.html#Poll10

[164] Post, D.E., Kendall, R.P.: Software project management and quality engineering practices for complex, coupled multiphysics, massively parallel computational simulations: Lessons learned from asci. International Journal of High Performance Computing Applications 18(4), 399–416 (2004)

[165] Pressman, R.S.: Software engineering: a practitioner's approach. Palgrave Macmillan (2005)

[166] Puolitaival, O.P.: Adapting model-based testing to agile context. VTT (2008)

[167] Rachatasumrit, N., Kim, M.: An empirical investigation into the impact of refactoring on regression testing. In: Software Maintenance (ICSM), 2012 28th IEEE International Conference on, pp. 357–366. IEEE (2012)

[168] Ramakrishnan, L., Gannon, D.: A survey of distributed workflow characteristics and resource requirements. Indiana University pp. 1–23 (2008)

[169] Remmel, H., Paech, B., Bastian, P., Engwer, C.: System testing a scientific framework using a regression-test environment. Computing in Science & Engineering 14(2), 38–45 (2012)

[170] Remmel, H., Paech, B., Engwer, C., Bastian, P.: Supporting the testing of scientific frameworks with software product line engineering: a proposed approach. In: Proceedings of the 4th International Workshop on Software Engineering for Computational Science and Engineering, pp. 10–18. ACM (2011)

[171] Reuillon, R., Chuffart, F., Leclaire, M., Faure, T., Dumoulin, N., Hill, D.: Declarative task delegation in openmole. In: High performance computing and simulation (hpcs), 2010 international conference on, pp. 55–62. IEEE (2010)

[172] Reuillon, R., Leclaire, M., Rey-Coyrehourcq, S.: Openmole, a workflow engine specifically tailored for the distributed exploration of simulation models. Future Generation Computer Systems 29(8), 1981–1990 (2013)

[173] Romain Reuillon Mathieu Leclaire, S.R.C.: Openmole, a workflow engine specifically tailored for the distributed exploration of simulation models. Future Generation Computer Systems 29(8), 1981 – 1990 (2013). URL http://www.openmole.org/files/FGCS2013.pdf

[174] Roy, C.: Practical software engineering strategies for scientific computing. In: Proceedings of the 19th AIAA Computational Fluid Dynamics Conference, pp. 1473–1485. Curran Associates, Inc, Red Hook, NY, USA (2009)

[175] Sanders, R.: The development and use of scientific software (2008)

[176] Sanders, R., Kelly, D.: The challenge of testing scientific software. In: Proceedings of the Conference for the Association for Software Testing, pp. 30–36 (2008)

[177] Sanders, R., Kelly, D.: Dealing with risk in scientific software development. IEEE software 25(4), 21 (2008)

[178] Saunders, M.H.: American national standards institute. Ph.D. thesis, National Institute of Standards and Technology (2017)

[179] Scacchi, W.: Process models in software engineering. Encyclopedia of software engineering (2001)

[180] Sciavicco, G., Juarez, J.M., Campos, M.: Quality checking of medical guidelines using interval temporal logics: A case-study. In: International Work-Conference on the Interplay Between Natural and Artificial Computation, pp. 158–167. Springer (2009)

[181] Segal, J.: Models of scientific software development. In: Proc. 2008 Workshop Software Eng. in Computational Science and Eng. (2008)

[182] Segal, J.: Scientists and software engineers: A tale of two cultures. In: Proceedings of the Psychology of Programming Interest Group, pp. 44–51. University of Lancaster, UK (2008)

[183] Segal, J., Morris, C.: Developing scientific software. IEEE Software 25(4), 18–20 (2008)

[184] Sengupta, B., Chandra, S., Sinha, V.: A research agenda for distributed software development. In: Proceedings of the 28th international conference on Software engineering, pp. 731–740. ACM (2006)

[185] Shrivastava, S.V., et al.: Distributed agile software development: A review. arXiv preprint arXiv:1006.1955 (2010)

[186] Siegel, S., Avrunin, G.: Verification of mpi-based software for scientific computation. In: S. Graf, L. Mounier (eds.) Model Checking Software, Lecture Notes in Computer Science, vol. 2989, pp. 286–303. Springer Berlin Heidelberg (2004). DOI 10.1007/978-3-540-24732-6_20. URL http://dx.doi.org/10.1007/978-3-540-24732-6_20

[187] Siegel, S., Rossi, L.: Analyzing blobflow: A case study using model checking to verify parallel scientific software. In: A. Lastovetsky, T. Kechadi, J. Dongarra (eds.) Recent Advances in Parallel Virtual Machine and Message Passing Interface, Lecture Notes in Computer Science, vol. 5205, pp. 274–282. Springer Berlin Heidelberg (2008). DOI 10.1007/978-3-540-87475-1_37. URL http://dx.doi.org/10.1007/978-3-540-87475-1_37

[188] Singh, Y., Kaur, A., Suri, B.: A hybrid approach for regression testing in interprocedural program. JIPS 6(1), 21–32 (2010)

[189] Skjellum, A., Bangalore, P., Gray, J., Bryant, B.: Reinventing explicit parallel programming for improved engineering of high performance computing software. In: Proceedings of the First International Workshop on Software Engineering for High Performance Computing System Applications, pp. 59–63. IEE, Edinburgh, Scotland, United Kingdom (2004)

[190] Sommerville, I.: Software Engineering. International computer science series. Addison-Wesley (2007). URL http://books.google.mk/books?id=B7idKfL0H64C

[191] Spangler, T.: Algorithms for grid graphs in the mapreduce model. Ph.D. thesis, University of Nebraska (2013)

[192] Srikanth, H., Williams, L., Osborne, J.: System test case prioritization of new and regression test cases. In: 2005 International Symposium on Empirical Software Engineering, 2005., pp. 10–pp. IEEE (2005)

[193] Srivastva, P.R., Kumar, K., Raghurama, G.: Test case prioritization based on requirements and risk factors. ACM SIGSOFT Software Engineering Notes 33(4), 7 (2008)

[194] Stackpole, C.S.: A Project Manager's Book of Forms: A Companion to the PMBOK Guide. John Wiley & Sons (2013)

[195] Stewart, J.M.: Python for scientists. Cambridge University Press (2014)

[196] Szalay, V., Czakó, G., Nagy, A., Furtenbacher, T., Császár, A.G.: On one-dimensional discrete variable representations with general basis functions. The Journal of chemical physics 119(20), 10,512–10,518 (2003)

[197] Tabaa, Y., Medouri, A.: Towards a next generation of scientific computing in the cloud. International Journal of Computer Science Issues(IJCSI) 9(6) (2012)

[198] Talby, D., Keren, A., Hazzan, O., Dubinsky, Y.: Agile software testing in a large-scale project. IEEE software 23(4), 30–37 (2006)

[199] Tan, W., Missier, P., Foster, I., Madduri, R., De Roure, D., Goble, C.: A comparison of using taverna and BPEL in building scientific workflows: the case of caGrid. Concurrency and Computation: Practice and Experience 22(9), 1098–1117 (2010)

[200] Thusoo, A., Sarma, J.S., Jain, N., Shao, Z., Chakka, P., Anthony, S., Liu, H., Wyckoff, P., Murthy, R.: Hive: A warehousing solution over a map-reduce framework. Proc. VLDB Endow. 2(2), 1626–1629 (2009). URL http://dl.acm.org/citation.cfm?id=1687553.1687609

[201] Tihana, G.G., Runeson, P., Darko, H.: Quantitative analysis of unit verification as predictor in large scale software enginering. Software Quality Journal (2015)

[202] Tossell, J.: Nuclear magnetic shieldings and molecular structure. NATO ASI series: Mathematical and physical sciences. Kluwer Academic Publishers (1993)

[203] Umarji, M., Seaman, C., Koru, A.G., Liu, H.: Software engineering education for bioinformatics. In: 2009 22nd Conference on Software Engineering Education and Training, pp. 216–223. IEEE (2009)

[204] Vagac, M., Kollar, J.: Improving program comprehension by automatic metamodel abstraction. Computer Science and Information Systems 9(1), 235–247 (2012)

[205] Vilkomir, S.A., Swain, W.T., Poore, J.H., Clarno, K.T.: Modeling input space for testing scientific computational software: A case study. In: Proceedings of the 8th international conference on Computational Science, pp. 291–300. Springer-Verlag, Berlin, Heidelberg (2008)

[206] Wang, G.: Evaluating mapreduce system performance: A simulation approach. Ph.D. thesis, Virginia Polytechnic Institute and State University (2012)

[207] White, T.: Hadoop: The Definitive Guide, 1st edn. O'Reilly Media, Inc. (2009)

[208] Wilensky, U.: {NetLogo} (1999)

[209] Wilson, G., Aruliah, D., Brown, C.T., Hong, N.P.C., Davis, M., Guy, R.T., Haddock, S.H., Huff, K.D., Mitchell, I.M., Plumbley, M.D., et al.: Best practices for scientific computing. PLoS Biol 12(1), e1001,745 (2014)

[210] Wilson, G., Aruliah, D.A., Brown, C.T., Chue Hong, N.P., Davis, M., Guy, R.T., Haddock, S.H.D., Huff, K., Mitchell, I.M., Plumbley, M., Waugh, B., White, E.P., Wilson, P.: Best Practices for Scientific Computing (2012). URL http://arxiv.org/abs/1210.0530

[211] Wilson, G.V.: Where's the real bottleneck in scientific computing? American Scientist 94(1), 5 (2006)

[212] Wolinski, K., Hinton, J.F., Pulay, P.: Efficient implementation of the gauge-independent atomic orbital method for nmr chemical shift calculations. Journal of the American Chemical Society 112(23), 8251–8260 (1990). DOI 10.1021/ja00179a005

[213] Wolstencroft, K., Haines, R., Fellows, D., Williams, A., Withers, D., Owen, S., Soiland-Reyes, S., Dunlop, I., Nenadic, A., Fisher, P., et al.: The taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud. Nucleic acids research p. gkt328 (2013)

[214] Wong, W.E., Horgan, J.R., London, S., Agrawal, H.: A study of effective regression testing in practice. In: Software Reliability Engineering, 1997. Proceedings., The Eighth International Symposium on, pp. 264–274. IEEE (1997)

[215] Woodcock, J., Larsen, P.G., Bicarregui, J., Fitzgerald, J.: Formal methods: Practice and experience. ACM Comput. Surv. 41(4), 19:1–19:36 (2009). DOI 10.1145/1592434.1592436. URL http://doi.acm.org/10.1145/1592434.1592436

[216] Yu, J., Buyya, R.: A taxonomy of scientific workflow systems for grid computing. SIGMOD Rec. 34(3), 44–49 (2005). DOI 10.1145/1084805.1084814. URL http://doi.acm.org/10.1145/1084805.1084814

[217] Zedan, H., Cau, A., Chen, Z., Yang, H.: Atom: An object-based formal method for real-time systems. Annals of Software Engineering 7(1-4), 235–256 (1999). DOI 10.1023/A:1018942406449. URL http://dx.doi.org/10.1023/A%3A1018942406449

[218] Zhang, C., Sterck, H., Aboulnaga, A., Djambazian, H., Sladek, R.: Case study of scientific data processing on a cloud using hadoop. In: D. Mewhort, N. Cann, G. Slater, T. Naughton (eds.) High Performance Computing Systems and Applications, Lecture Notes in Computer Science, vol. 5976, pp. 400–415. Springer Berlin Heidelberg (2010). URL http://dx.doi.org/10.1007/978-3-642-12659-8_29

[219] Zhang, J., Cheng, B.H.C.: Using temporal logic to specify adaptive program semantics. Journal of Systems and Software 79(10), 1361–1369 (2006). URL http://dblp.uni-trier.de/db/journals/jss/jss79.html#ZhangC06

[220] Zhao, Y., Raicu, I., Foster, I.: Scientific workflow systems for 21st century, new bottle or new wine? In: Proceedings of the 2008 IEEE Congress on Services - Part I, SERVICES '08, pp. 467–471. IEEE Computer Society, Washington, DC, USA (2008). DOI 10.1109/SERVICES-1.2008.79. URL http://dx.doi.org/10.1109/SERVICES-1.2008.79

[221] Zheng, B.: Documentation driven testing of scientific computing software. Master Thesis, McMaster University (2009). [Online]. Available: http://digitalcommons.mcmaster.ca/cgi/viewcontent.cgi?article=5421&context=opendissertations (current November 2012)

[222] Ziff Davis Publishing Holdings, I.: Pc magazine (1995). URL http://www.pcmag.com/encyclopedia/term/50872/scientific-application