

Zoran Budimac, Tihana Galinac Grbac (Eds.)

Third Workshop on Software Quality Analysis, Monitoring, Improvement and Applications

SQAMIA 2014

Lovran, Croatia, September 19–22, 2014

Proceedings

Department of Mathematics and Informatics
Faculty of Sciences, University of Novi Sad, Serbia
2014

Volume Editors

Zoran Budimac
University of Novi Sad
Faculty of Sciences, Department of Mathematics and Informatics
Trg Dositeja Obradovića 4, 21000 Novi Sad, Serbia
E-mail: zjb@dmi.uns.ac.rs

Tihana Galinac Grbac
University of Rijeka
Faculty of Engineering, Department of Computer Engineering
Vukovarska 58, 51000 Rijeka, Croatia
E-mail: tgalinac@riteh.hr

Papers are copyrighted © 2014 by the authors. Copying permitted only for private and academic purposes. This volume is published and copyrighted by its editors. Proceedings also appeared in ISBN 978-86-7031-374-3, Faculty of Sciences, University of Novi Sad. The contents of the published papers express the opinions of their respective authors, not the volume publisher.

ISBN: 978-86-7031-374-3

Preface

This volume contains papers presented at the Third Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications (SQAMIA 2014). SQAMIA 2014 was held during September 19-22, 2014, in Lovran, Croatia.

SQAMIA 2014 is a continuation of two successful events held in 2012 and 2013 in Novi Sad, Serbia. The first SQAMIA workshop was organized within the 5th Balkan Conference in Informatics (BCI 2012) in Novi Sad. In 2013, SQAMIA became a standalone event in intention to become traditional meeting of the scientists and practitioners in the field of software quality.

The main objective of SQAMIA workshop series is to provide a forum for presentation, discussion and dissemination of the scientific findings in the area of software quality, and to promote and improve interaction and cooperation between scientists and young researchers from the region and beyond. SQAMIA workshops traditionally welcome position papers, papers describing the work-in-progress, tool demonstration papers, technical reports, or other papers that would provoke discussions.

The SQAMIA 2014 workshop consisted of regular sessions with technical contributions reviewed and selected by an international program committee, as well as of invited talks presented by leading scientists in the research areas of the workshop.

In total, 14 papers were accepted and published in this proceedings volume. All published papers were double reviewed, and some papers received the attention of more than two reviewers. We would like to use this opportunity to thank all PC members and the external reviewers for submitting careful and timely opinions on submitted papers.

Also, we gratefully acknowledge the program co-chairs, Marjan Heričko (Slovenia), Zoltán Horváth (Hungary), Mirjana Ivanović (Serbia), and Hannu Jaakkola (Finland), for helping to greatly improve the quality of the workshop.

We extend special thanks to the SQAMIA 2014 Organizing Committee of the Department of Computer Engineering, Faculty of Engineering, University of Rijeka, Croatia and Department of Mathematics and Informatics, Faculty of Sciences, University of Novi Sad, Serbia.

Finally, we thank our sponsors for supporting the organization of this event. We are especially thankful for financial support by the EU COST Action IC1202: Timing Analysis on Code-Level (TACLe) and to University of Rijeka providing support by research Grant 13.09.2.2.16.

We are also grateful to the Remisens Family Hotel Excelsior for supporting the workshop by providing the lecture hall and equipment. We also thank the Lovran Municipal Tourism Board for making the participants bags heavier.

And last, but not least, we thank all the participants of SQAMIA 2014 for having made all work that went into SQAMIA 2014 worthwhile.

September 2014

*Zoran Budimac
Tihana Galinac Grbac*

Workshop Organization

General Chair

Tihana Galinac Grbac (*Univ. of Rijeka, Croatia*)

Program Chair

Zoran Budimac (*Univ. of Novi Sad, Serbia*)

Program Co-Chairs

Tihana Galinac Grbac (*Univ. of Rijeka, Croatia*)

Marjan Heričko (*Univ. of Maribor, Slovenia*)

Zoltán Horváth (*Eötvös Loránd Univ., Budapest, Hungary*)

Mirjana Ivanović (*Univ. of Novi Sad, Serbia*)

Hannu Jaakkola (*Tampere Univ. of Technology, Pori, Finland*)

Program Committee

Zoran Budimac (*Univ. of Novi Sad, Serbia*)

Tihana Galinac Grbac (*Univ. of Rijeka, Croatia*)

Marjan Heričko (*Univ. of Maribor, Slovenia*)

Zoltán Horváth (*Eötvös Loránd Univ., Budapest, Hungary*)

Mirjana Ivanović (*Univ. of Novi Sad, Serbia*)

Hannu Jaakkola (*Tampere Univ. of Technology, Pori, Finland*)

Vladimir Kurbalija (*Univ. of Novi Sad, Serbia*)

Anastas Mishev (*Univ. of Ss. Cyril and Methodius, Skopje, FYR Macedonia*)

Sanjay Misra (*Atilim Univ., Ankara, Turkey*)

Vili Podgorelec (*Univ. of Maribor, Slovenia*)

Zoltán Porkoláb (*Eötvös Loránd Univ., Budapest, Hungary*)

Valentino Vranić (*Slovak Univ. of Technology, Bratislava, Slovakia*)

Additional Reviewers

Jean Petrić (*Univ. of Rijeka, Croatia*)

Harri Keto (*Tampere Univ. of Technology, Pori, Finland*)

Organizing Committee

Goran Mauša, Chair (*Univ. of Rijeka, Croatia*)

Jean Petrić (*Univ. of Rijeka, Croatia*)

Gordana Rakić (*Univ. of Novi Sad, Serbia*)

Miloš Savić (*Univ. of Novi Sad, Serbia*)

Organizing Institution

Department of Computer Engineering, Faculty of Engineering, University of Rijeka, Croatia

Sponsoring Institutions of SQAMIA 2014

SQAMIA 2014 was partially financially supported by:

EU COST Action IC1202 *Timing Analysis on Code-Level* (TACLe)

University of Rijeka (research grant 13.09.2.2.16)

Table of Contents

◦ Criteria for Selecting Mobile Application Testing Tools	1
<i>Boštjan Arzenšek, Marjan Heričko</i>	
◦ Attribute-based Checking of C++ Move Semantics	9
<i>Áron Baráth, Zoltán Porkoláb</i>	
◦ Clone Wars	15
<i>Viktória Fördős, Melinda Tóth, Tamás Kozsik</i>	
◦ Information and Information Security	23
<i>Jaak Henno</i>	
◦ Formal Specification of Scientific Applications Using Interval Temporal Logic	29
<i>Bojana Koteska, Ljupco Pejov, Anastas Mishev</i>	
◦ Towards a Framework for Usability Testing of Interactive Touchless Applications	39
<i>Saša Kuhar, Kristjan Košič</i>	
◦ Techniques for Bug-Code Linking	47
<i>Goran Mauša, Paolo Perković, Tihana Galinac Grbac, Ivan Štajduhar</i>	
◦ Processing and Data Collection of Program Structures in Open Source Repositories	57
<i>Jean Petrić, Tihana Galinac Grbac, Mario Dubravac</i>	
◦ Tool for Testing Bad Student Programs	67
<i>Ivan Pribela, Doni Pracner, Zoran Budimac</i>	
◦ Toward Language Independent Worst-Case Execution Time Calculation	75
<i>Gordana Rakić, Zoran Budimac</i>	
◦ Graph Clustering Evaluation Metrics as Software Metrics	81
<i>Miloš Savić, Mirjana Ivanović</i>	
◦ Approaches for Test Case Generation from UML Diagrams	91
<i>Tina Schweighofer, Marjan Heričko</i>	
◦ Prototype System for Improving Manually Collected Data Quality	99
<i>Jari Soini, Pekka Sillberg, Petri Rantanen</i>	
◦ The Effect of Educating Users on Passwords: a Preliminary Study	107
<i>Viktor Taneski, Boštjan Brumen, Marjan Heričko</i>	

Criteria for Selecting Mobile Application Testing Tools

BOŠTJAN ARZENŠEK and MARJAN HERIČKO, University of Maribor

The importance of software testing has been discussed and proven in many articles and in existing research. Software testing plays a key role in providing quality software solutions and services. Standard testing approaches and methodologies were adequate until the arrival of mobile technologies. With mobile technologies, the testing process was forced to change in the face of significant challenges, the most important one being mobility. Mobility provides a pallet of challenges that are unique and demand new testing approaches and methodologies in software testing. The identification of challenges and issues has helped the development of the mobile software testing process and tools. With a wide range of new testing tools, testers face a new challenge in selecting the right tool and methodology for testing mobile applications. In this paper, we will present criteria for selecting mobile application testing tools based on identified challenges and issues, testing approaches and strategies. We will provide a proposal for a simpler and quicker way of selecting the appropriate tool for testing mobile applications.

General Terms: Mobile applications testing

Additional Key Words and Phrases: testing, mobile applications, mobile technologies

1. INTRODUCTION

The increased use and the rapid development of mobile devices and technology is a clear sign of future trends in ICT. Not only do the number of mobile users increase daily, the same thing is occurring in the market for tablet PCs. According to Gartner, the number of purchased tablets is going to surpass the number of desktop-based computers in the beginning of the year 2015 [Rivera and Van der Meulen, 2013]. More and more people are using mobile technologies in their everyday lives for interaction, entertainment, business and more. With mobile applications we can extend the usability of mobile devices even further. A mobile application is an application that runs on a mobile device and is context aware. This means that the application is aware of the computing environment in which it runs and can adapt/react according to its current context [Muccini, 2012]. Context-awareness is just one of the many challenges in mobile application testing, which demands new methods and testing approaches. There are many challenges in mobile software testing, which by definition [Gao et al. 2013] means: “testing activities for mobile-based applications and mobile web applications on mobile devices using well-defined software test methods and tools to ensure the quality in mobile service functions, behaviors, performance and QoS, as well as mobile features, such as mobility, usability, inter-operability, mobile connectivity, security and privacy.” Mobile applications that are free of faults and errors provide a better user experience, which has a direct impact on the business success of the application. Users grade the quality of the mobile application based on their user experience. Unfortunately, many new users choose applications based on previous reviews and grades. Therefore, old errors and faults, or a poor user experience in an otherwise working application can lead to the business failure of the application.

In this paper we will present the criteria for selecting mobile application tools based on the identified challenges in mobile testing, testing approaches and strategies. First, in Section 2, we will present the main challenges and issues in mobile software testing. In Section 3, we will present the four testing approaches that have been identified by [Gao et al. 2013]. In Section 4, we will present and analyze related work on mobile testing tools, which provides the basis for our criteria definition. Our main focus is to present the criteria definition process and to extend the criteria for selecting mobile testing tools that have been identified by [Gao et al. 2013]. The idea is to have a selection of criteria that can be used for the characterization of a variety of mobile testing tools. Once we have the mobile testing tools characterized

Author's address: B. Arzenšek, Faculty of Electrical Engineering and Computer Science, University of Maribor, Smetanova 17, 2000 Maribor, Slovenia; email: bostjan.arzensek@um.si; M. Heričko, Faculty of Electrical Engineering and Computer Science, University of Maribor, Smetanova 17, 2000 Maribor, Slovenia; email: marjan.hericko@um.si.

Copyright © by the paper's authors. Copying permitted only for private and academic purposes.

In: Z. Budimac, T. Galinac Grbac (eds.): Proceedings of the 3rd Workshop on Software Quality, Analysis, Monitoring, Improvement, and Applications (SQAMIA), Lovran, Croatia, 19.-22.9.2014, published at <http://ceur-ws.org>

properly we can make a quicker and more effective selection. In the Discussion, we will comment on our findings.

2. CHALLENGES OF MOBILE TESTING

There are many challenges in mobile software testing due to the nature of the environment in which the mobile applications are running. Based on the definition by [Muccini 2012]: “A mobile application is an application that runs on mobile device with limited resources. It uses the data from the surrounding environment in which the mobile device is in and/or from user actions, producing context-based output.” Mobile applications differ from traditional desktop application in many ways. In this section we will describe the main challenges in mobile software testing and why the traditional tools and methodology of software testing are not adequate.

2.1 Mobile Connectivity

One of the more important challenges in mobile software testing is the connectivity of mobile devices with various mobile networks and devices. Unlike desktop applications, which use fixed network connections, mobile applications connect to mobile networks, which can vary in speed, security and reliability [Kirubakaran and Karthikeyani 2013]. Usually the types of mobile networks are 2G, 3G, 4G and various wireless networks. Mobile applications rely heavily on mobile networks, which is why the challenge of mobility can have an impact on: reliability, performance, security and the correct operation of the application and/or its functionalities [Muccini 2012]. The nature of the challenge demands testing in different environments. The mobile applications are tested in:

- environment with a constant connection to the mobile network,
- environment with a variable connection to the mobile network and
- environment without a connection.

Based on the difficulties and requirements of the testing procedures, different testing approaches are recommended [Tharakan and Jacob 2012]. Because the application’s reliability, performance, security and correct functioning strongly depend on the available connection type, functional and extra functional testing has to be performed in different connectivity scenarios and networks [Muccini 2012].

2.2 Resource constraints

Mobile applications use the resources of mobile devices, which are very limited. Despite the rapid development of mobile devices, it is important that the consumption of resources is monitored and controlled at all times [Muccini 2012]. The resources of mobile devices include: the central processing unit, RAM, memory, touch screen, battery, as well as different modules and sensors. During the testing process, we focused on the central processing unit, RAM and memory. Because the battery and the screen constitute a different set of challenges, we treated them individually. The central processing unit, RAM and memory are components of the SoC (System-on-a-Chip) which includes other controllers and components that form a complete system [Yeap, 2013].

The excessive use of resources can reduce the performance of mobile devices and can cause malfunctions in the mobile application. During the testing process the consumption of resources must be constantly monitored.

2.3 Autonomy

Mobile devices need energy to run. The use of mobile devices depends on battery capacity and the way the device is used. All the device’s resources and activities use energy but not equally. GPS sensors, data transfer and video editing are activities that use more energy than others [Tharakan and Jacob 2012]. These activities use multiple device resources or require a continuous data connection, which is the main reason for higher energy consumption. Different activities have a different impact on autonomy and during the testing process all have to be monitored [Muccini 2012].

2.4 Diversity of user interfaces

Mobile operating systems have different user interfaces, which are defined by rules and guidelines. The use and layout of elements is checked in the verification process when publishing the mobile applications on the markets. Non-compliance with rules and guidelines can delay the publishing process, increase the cost of development and testing. Different screen sizes can also have an impact on the look and usability of the mobile application. Different mobile devices can react differently to the same application code, which must be tested with GUI testing [Muccini 2012].

2.5 Context awareness

Context is by definition [Abowd et al. 1999] any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the users and applications themselves.

Context can limit or extend the operation of mobile applications or its functionalities with data from the environment in which it is in. Mobile applications can be in different contexts with different data. This creates a unique challenges in the testing process [Schulte and Majchrzak, 2012].

Context aware mobile applications adapt and evolve based on the data obtained from the environment. This evolution can happen in real time without interrupting or stopping the operation of mobile applications. Unfortunately, this can lead to unexpected and unplanned changes in the mobile application's operations. The reliability of a mobile application depends on the management of context adaption. To insure the correctness of applications operation, context-specific test selection techniques and coverage criteria have to be produced [Muccini 2012].

2.6 Diversity of mobile devices

There are many different mobile devices, made by different vendors, which have different hardware and software settings. The number of variations is even larger if we add all the devices that have a modified mobile operating system. The vendors modify the operating system to create a better user experience for the user, or increase the functionalities of a device. Due to these variations, mobile applications can run and behave differently [Muccini 2012]. The diversity of mobile devices can also increase the costs and duration of the testing process. If we would want to test across all devices, the buying and maintaining costs of mobile devices would be enormous. If we take into account the time spent for testing, the complexity of the challenge increases. Testing techniques that maximize the diversity coverage while minimizing the devices being tested need to be devised [Muccini 2012].

2.7 User experience

The user experience includes the user's perceptions and feelings before, during and after the interaction with the mobile application. Often, users assess the application based on their user experience, therefore the appropriate user experience is critical for the success of the application. The adequacy of the user experience cannot be directly tested because of the subjective nature of the entire process. But we could check the correctness of individual segments and determine compliance with good practices [Knab 2012].

The adequacy of the user experience includes verification of elements and activities in the areas of design graphical user interfaces, interaction and usability of the application itself.

The design of the graphical interface is evaluated based on the proper and logical use of layouts, navigation between screens, layout of graphical elements, fonts and text [Knab 2012].

2.8 Touch screens

Mobile devices use touch screens as the primary means of interacting with the user. Touch screens enable the display and input of data as individual values or as a group of data. The user activates the interaction with a touch of the screen, which can be a single touch or a multi touch interaction. There are also gestures that include different sequences and combinations of touches. Gestures allow additional functionalities in the mobile application's operation, which creates new challenges in the testing process. Touch screens are tested based on the correctness of the displayed data and the responsiveness

[Kirubakaran and Karthikeyani 2013]. The responsiveness of the touch screen represents the elapsed time between the touch of the screen and the moment, when the touch is recognized, which triggers an update on the screen [Agawi App Glimpse n.d.]. The responsiveness of the touch screen is also dependent on the mobile device's resources.

2.9 New programming languages and mobile operating systems

Programming languages for mobile applications have been designed to support mobility, resource management and new graphical user interfaces. Traditional testing techniques do not take into account the operation of programming languages in mobile operating systems, so they need to be adjusted accordingly. To analyze the code it is necessary to be aware of the specifics of the programming languages and how they operate [Kirubakaran and Karthikeyani 2013].

Mobile operating systems are new and still only partially reliable. Various reliability issues and defaults are corrected in new versions of the operating system which are frequent but not always backward compatible [Muccini 2012]. For example, mobile devices with Microsoft mobile operating system Windows Phone 7 were not updatable with the new release of Windows Phone 8 [Hamblen 2009].

3. MOBILE TESTING APPROACHES

Based on [Gao et al. 2013] there are four testing approaches in mobile testing. These approaches are:

- emulation-based testing,
- device-based testing,
- cloud testing and
- crowd-based testing.

Each of them is designed to handle challenges in mobile testing, identified in the previous section, but none of them can handle them all. It is important to select the correct approach based on the functionalities of the mobile application and the challenges that they provide. Each approach has its features and limitations, which have to be identified before the selection is made.

3.1 Emulator-based testing

The emulation-based testing approach involves using a mobile device emulator, which creates a virtual machine version of a mobile device for inspection on a personal computer. It is often included with a mobile platform's software development kit. It is relatively inexpensive because no testing laboratory is needed and no physical devices have to be purchased or rented, but it can only be used to assess system functionality within a very limited context. Although this approach is low-cost, it has several limitations – for example, it has difficulty validating a full set of gestures because most emulators support very limited gestures and device-specific functions. Another challenge is its limited scale for testing QoS. To overcome these problems, a simulation-based approach can create a mobile test simulator to mimic various mobile client operations and support more than one mobile client. However, even this workaround has its limitations in validating device-specific mobile service functions. In addition, it is impossible to deal with diverse devices and mobile platforms because emulators are usually based on a specific device platform [Gao et al. 2014].

3.2 Device-based testing

The device-based testing approach requires setting up a testing laboratory and purchasing real mobile devices, which is more costly than emulation-based approaches but can verify device based functions, behaviors, and QoS parameters that other approaches cannot. In addition, it also has the advantage of being able to validate its underlying mobile networks via reconfiguration and selections in a testing environment. One of the major challenges with this approach is the problem it has in coping with rapid changes in mobile devices and platforms. Another challenge is its limitations related to system QoS because large-scale testing require many mobile devices, which is usually impossible for enterprises [Gao et al. 2014].

3.3 Cloud testing

This approach, based on testing through the cloud, is typically supported by testing vendors. The basic idea is to build a mobile device cloud that can support testing services on a large scale. This approach addresses the significant increase in demand for mobile testing services by using the pay-as-you-go business model. It also allows different mobile users to provision their required testing environment via a rental service model. Compared with other approaches, this can be more cost-effective than device-based testing for large-scale applications, and is much more effective for supporting diverse testing activities on mobile devices.

3.4 Crowd-based testing

The crowd-based testing approach involves using freelance or contracted testing engineers or a community of end users such as uTest (www.utest.com), along with a crowd-based testing infrastructure and service management server to support diverse users. Currently, a service vendor supports primitive test management, a testing service, and bug reporting. Most mobile test operations are managed in an *ad hoc* way with very limited mobile test automation tools. This approach offers the benefit of in-the-wild testing without the need to invest in a laboratory or purchase or rent devices, but at the risk of low testing quality and an uncertain validation schedule.

4. RELATED WORK

Recent studies on mobile testing tools primarily focus on GUI-based testing, test automation and white-box testing. There are different studies that present solutions for different testing approaches and strategies, which has created numerous tools for mobile testing [Gao et al. 2014]. If we want to make an efficient selection, we must first characterise these tools and then evaluate them. The process of evaluation has many challenges. One of the reasons for this is the lack of standards, test models and coverage criteria that address the distinct requirements of mobile application testing [Gao et al. 2014]. One of the challenges is also defining the criteria with which these tools are evaluated. In [Gao et al. 2014] the testing tools are compared based on different criteria, which are listed in Table I. This comparison is a good example of a possible mobile testing tool characterization.

Table I. Criteria defined in a mobile testing tools comparison [Gao et al. 2014]

Criteria	Values
Testing strategy	GUI-based function testing
	Performance testing
	Load testing
Mobile testing tool platform	Linux
	Windows
	Mac
Mobile application platform	Android OS
	iOS
	Windows OS
Mobile application type	Native apps
	Web apps
Testing approaches	Emulation-based testing
	Device-based testing
Test properties	Supported script language
	Record and Play
License	Open source
	Subscription

The criteria are defined based on the current functionalities of the testing tools and the identified testing strategies and approaches.

Although the provided criteria allow a basic characterization of the testing tools, we believe that the criteria should be extended and created based on challenges in mobile testing. With a more extended definition of criteria, the selection process could be faster and more efficient.

5. CRITERIA FOR SELECTING MOBILE APPLICATION TESTING TOOLS

Based on related work described in the previous section and the identified challenges, we propose a definition of criteria that is based on these challenges and the testing strategies. Our goal is to create more detailed criteria for selecting the mobile testing tool.

The definition process consists of three phases and is illustrated in Figure 1.

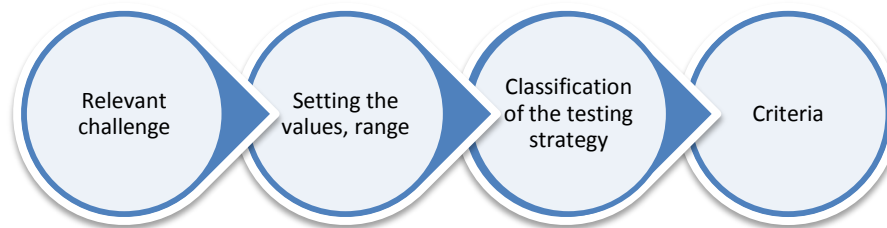


Fig. 1. The definition process of a criteria

The process begins with the selection of the mobile testing challenge, which we have identified in Section 2. The relevant challenge is analyzed from the perspective of the mobile application and based on this result, the values or the range are set. Values and range can differ depending on the type of the challenge. For example, the values for screen dimensions are small, normal, large and extra large, where all the sizes are based on [Google 2014], and values for the challenge of Bluetooth connectivity, where the values are only enabled or not enabled. After the values have been set, it is important to classify the upcoming criteria into a correct strategy. The testing strategy defines the testing activities and the goals of the testing process. Finally, after the testing strategy is defined, the criteria is created. The defined criteria is used to characterize the testing tools. Before the evaluation of the testing tools can be done, the selection of criteria must be set. The selection criteria are set based on the mobile application functionalities and on the selected testing approach. When the testing tools are selected and the criteria are set, the evaluation process can be begin.

Based on the process of creating criteria we propose a list of criteria that are defined based on the challenges described in section 2. The proposed criteria are shown in Table II.

Table II. Proposed criteria that are defined based on the challenges

Challenge	Properties	Values, range	Testing strategy	Supported feature
Mobile connectivity	Mobile network	Constant, partial, none	Functional testing	Supports changing the consistency of the mobile network
	Data transfer speed	Range of speeds (2G, 3G and 4G)	Connectivity testing	Supports changing or limiting the data transfer speed
	Bluetooth	Enabled, not enabled	Functional testing	Supports Bluetooth connectivity
	NFC	Enabled, not enabled	Functional testing	Supports NFC connectivity
	Wi-Fi	Enabled, not enabled	Functional testing	Supports Wi-Fi connectivity

	Wi-Fi Direct	Enabled, not enabled	Functional testing	Supports Wi-Fi Direct connectivity
Resource constraints	CPU	1 core, 2 core, 4 cores	Performance testing	Supports changing or limiting the operation of CPU cores
	CPU Speed	Range of speeds (1Mhz – 2500Mhz)	Performance testing	Supports changing or limiting the operation of CPU speed
	RAM	Range of values (16Mb,32Mb,64Mb, 128Mb,256Mb,512Mb, 768Mb,1Gb,2Gb,3Gb and 4Gb)	Performance testing	Supports changing or limiting the amount of RAM
	Memory	Range of values (16Mb,32Mb,64Mb, 128Mb,256Mb,512Mb, 768Mb,1Gb,2Gb,3Gb,4Gb,8 Gb,16Gb,32Gb, 64Gb,128Gb)	Performance testing	Supports changing or limiting the amount of memory
Autonomy	Consumption	Percentage of the total battery capacity	Load testing	Supports monitoring the battery consumption
	Duration	Time of the total battery capacity	Load testing	Supports monitoring the battery duration
Diversity of user interface	Guideline checker	Mobile platform specific Rules and guidelines	Usability testing	GUI guideline checker
Context awareness	GPS	Simulated, real data, not enabled	Functional testing	Simulate data from the GPS
	Neighbor devices	Simulated, real data, not enabled	Functional testing	Simulate data from the neighbor device
	Altitude	Simulated, real data, not enabled	Functional testing	Simulate data from the barometer
	Brightness	Simulated, real data, not enabled	Functional testing	Simulate data from the light sensor
	Temperature	Simulated, real data, not enabled	Functional testing	Simulate data from the temperature sensor
	Context awareness	Simulated, real data, not enabled	Functional testing	Simulate data from the environment and the user
	Context adaption	Enabled, not enabled	Functional testing	Enables data input from the context in real time
Diversity of mobile devices	Vendor and model	Enabled, not enabled	Functional testing	Simulation of a specific mobile device
	Operating system	Android, iOS, BlackBerry, Windows Phone 7 and 8	Functional testing	Supports changing mobile device platform
	Operating system versions	Enabled, not enabled	Functional testing	Supports changing mobile device platform to different versions
	Screen dimensions	Small (at least 426dp x 320dp), normal (at least 470dp x 320dp), large screen (at least 640dp x 480dp), extra large screen (at least 960dp x 720dp)	Usability testing	Supports changing screen size
User experience	Layout checker	Enabled, not enabled	Usability testing	Checks the use of layouts for specific mobile operating systems
	Text visibility	Percentage of the characters displayed based on the total number of characters	Usability testing	Checks text visibility
	Text grammar	Supported, not supported	Usability testing	Supports internationalization
	Notifications	Enabled, not enabled	Usability testing	Supports notification management
	Interruptions	Enabled, not enabled	Usability testing	Supports interruptions

				management
Touch screens	Responsiveness	Enabled to measure, not enabled to measure	Usability testing	Supports measuring the responsiveness of the screen
	Gestured	Enabled, not enabled	Usability testing	Supports gesture recognition
	Multi touch	Enabled, not enabled	Usability testing	Supports multi touch recognition

5. DISCUSSION

The proposed criteria and its values were set based on the defined challenges in mobile testing. The challenges in mobile software testing vary from traditional ones because of mobility. Mobility has changed the operation of applications, devices and our interaction with these devices. Consequently, mobility has changed the software testing process, which is nevertheless still evolving. Because of this, there is a possibility that the proposed criteria can change, adapt or possibly be removed from the list. Future research will confirm or refute this claim. Also, in the future we expect the development of new tools and techniques that will enable more effective mobile testing. New tools will enable more detail testing, simulation and better approaches for testers. This also presents a potential business opportunity in the field of mobile testing tools, test automation and in the increasingly popular cloud-based testing environment.

As previously mentioned, the future trend in ICT is mobility, which also applies for mobile testing. The world of mobile devices is rapidly developing, which requires rapid development in mobile testing.

REFERENCES

- Abowd, G., Dey, A., & Brown, P. (1999). Towards a better understanding of context and context-awareness. Agawi App Glimpse. (n.d.). TouchMarks I: Smartphone Touchscreen Latencies. Retrieved from <http://appglimpse.com/blog/touchmarks-i-smart-phone-touch-screen-latencies/>
- Gao, J., Bai, X., Tsai, W., & Uehara, T. (2013). Mobile Application Testing – Research , Practice , Issues and Needs Testing , Requirements and Features, 4(1).
- Gao, J., Jose, S., Tsai, W., & Uehara, T. (2014). Mobile application testing: A tutorial, 46–55.
- Google. (2014). Supporting Multiple Screens. Retrieved from http://developer.android.com/guide/practices/screens_support.html
- Hamblen, M. (2009). Ballmer: We “screwed up with Windows Mobile.” Retrieved from <http://news.idg.no/cw/art.cfm?id=F2F7C35E-1A64-67EA-E4BC04F120F0B898>
- Kirubakaran, B., & Karthikeyani, V. (2013). Mobile Application Testing – Challenges and Solution Approach through Automation.
- Knab, K. (2012). Main issues in mobile app testing. Testing Experience, 19.
- Muccini, H. (2012). Software testing of mobile applications: Challenges and future research directions, 29–35.
- Rivera, J., & Van der Meulen, R. (2013). Gartner Says Worldwide PC, Tablet and Mobile Phone Combined Shipments to Reach 2.4 Billion Units in 2013. Retrieved from <http://www.gartner.com/newsroom/id/2408515>
- Schulte, M., & A. Majchrzak, T. (2012). Context-Dependent Testing of Apps Applications. Testing Experience.
- Tharakan, M., & Jacob, J. (2012). Roadblocks and their workaround while testing Mobile Applications. Testing Experience, 19.
- Yeap, G. (2013). Smart mobile SoCs driving the semiconductor industry: technology trend, challenges and opportunities. IEDM Technical Digest, 16–23. Retrieved from <http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Smart+Mobile+SoCs+Driving+the+Semiconductor+Industry+:+Technology+Trend+,+Challenges+and+Opportunities#0>

Attribute-based Checking of C++ Move Semantics

ÁRON BARÁTH and ZOLTÁN PORKOLÁB, Eötvös Loránd University

Worst-case execution time (WCET) analysis is an important research area in various application areas, like real-time and embedded programming as well as a wide range of other high performance applications. However, WCET analysis is a complex and difficult area, because the processor's cache operations, interrupt routines and the operating system impact the execution time. Also, it is very difficult to replicate a specified scenario. Measuring the worst-case execution time as an absolute quantity therefore is still an unsolved problem for the most popular programming languages. In the same time, WCET calculation is also important on the source code level. When evaluating a library which can be compiled on different platforms with different compilers we are interested not in absolute or approximated time but rather certain elementary statements, e.g. number of (expensive) copy instructions of composite types.

In this paper we introduce a new approach of worst-case execution time investigation, which operates at language source level and targets the move semantics of the C++ programming language. The reason of such high-level worst-case execution time investigation is its platform and compiler independence: the various compiler optimizations will less impact the results. The unnecessary expensive copies of C++ objects – like copying containers instead of using move semantics – determine the main time characteristics of the programs. We detect such copies occurring in operations marked as *move operations*, i.e. intended not containing expensive actions.

We implemented a tool prototype to detect copy/move semantics errors in C++ programs. Move operations are marked with generalized attributes – a new feature introduced to C++11 standard. Our prototype is using the open source LLVM/Clang parser infrastructure, therefore highly portable.

Categories and Subject Descriptors: D.3.3 [Programming Languages] Language Constructs and Features; D.2.4 [Software Engineering] Software/Program Verification

Additional Key Words and Phrases: Programming languages, Execution time, Worst-case execution time estimation, C++, C++ generalized attributes, C++ move semantics

1. INTRODUCTION

In this paper we introduce a new approach of worst-case execution time investigation for C++ programs. Worst-case execution time estimation is important in different application areas, for example in embedded programming, and time critical, high responsive server programs. In these areas the "buy a better hardware" philosophy is not an option. The final code must be reviewed, tests and validations must be applied before the program being used. The validation process may contain worst-case execution time estimation (WCET) or measurement. Note that, such measurement are very hard to complete.

The WCET estimation can be defined in various ways, depending on which program features should be estimated. The most obvious but also the most difficult approach is the quantitative *time* estimation (e.g. in seconds) [Wilhelm et al. 2008; Huynh et al. 2011]. Such analysis is a complex and difficult area, because of the processor's cache operations, interrupt routines and the operating system impact the execution time. Also, it is very difficult to replicate a specified scenario. Measuring the worst-case

Author's address: Á. Baráth and Z. Porkoláb, Department of Programming Languages and Compilers, Faculty of Informatics, Eötvös Loránd University, Pázmány Péter sétány 1/C, H-1117 Budapest, Hungary; email: {baratharon, gsd}@caesar.elte.hu

Copyright © by the paper's authors. Copying permitted only for private and academic purposes.

In: Z. Budimac, T. Galinac Grbac (eds.): Proceedings of the 3rd Workshop on Software Quality, Analysis, Monitoring, Improvement, and Applications (SQAMIA), Lovran, Croatia, 19.-22.9.2014, published at <http://ceur-ws.org>.

execution time as an absolute quantity therefore is still an unsolved problem for the most popular programming languages, like C++.

In the other hand, we can make an estimation of the number of executed instructions – this method will ignore any cache accesses and interruptions, but still enough low level. Also, on RISC processors the estimation can be converted into an approximated time. However, these methods must be done for every binary versions and architectures, because the different compilers and optimization strategies.

In the same time, WCET calculation is also important on the source code level. When evaluating a library which can be compiled on different platforms with different compilers we are interested not in absolute or approximated time but rather certain elementary statements, e.g. number of (expensive) copy instructions of certain types.

This problem led to us the high-level WCET investigation. It allows us to process the code at the language source level, which is architecture independent. At the language level the number of assignments, comparisons, and copies can be measured. These discrete values can be capped by a predefined limit and can be checked by static analysis tools. Also, using such tools serious programming errors can be detected, what none of the low-level WCET estimations can reveal.

In this paper, we introduce such a tool for C++ programs, to detect copy/move semantics errors in order to ensure the pertinence of the program. Moreover, the number of copies can be checked, because the unnecessary copies can slow down the code as well.

This paper is organized as follows: in Section 2 we present the necessity of the move semantics in C++, and we present use cases to introduce the move semantics with related code snippets. In Section 3 we presented a new C++11 feature, called generalized attributes, and we used this mechanism to annotate the source code with expected semantics informations. Also, we introduced our Clang-based tool to check semantics and display error messages to the mistakes. Finally, in Section 4 we present our future plans with our tool.

2. C++ MOVE SEMANTICS

The C and C++ languages have value semantics [Stroustrup 1994]. When we use assignment, we copy raw bytes by default. This behavior can cause serious performance issues. In case of concatenating multiple arrays in one assignment, the cost of the temporary objects are high. The used `new` and `delete` operators, the overhead of the extra loops and memory accesses, and the amount of the allocated memory are costly operations. Todd Veldhuizen investigated this issue, and suggested C++ template metaprogramming [Alexandrescu 2001; Czarnecki and Eisenecker 2000; Sinkovics and Porkoláb 2012; Veldhuizen 1995b] and expression templates [Veldhuizen 1995a] as a solution. The base idea is to avoid the creation of temporaries, but “steal” the resources of the operands. Such operation can be implemented library-based [D. Abrahams 2004] almost trivial with overloading, but to distinguish non-destroyable objects from those which can be “wasted” language support is required. Therefore C++11 standard has introduced a new reference type: the *rvalue* reference [Standard 2011; str]. In the source code, the rvalue references have a new syntax: `&&`. Using this syntax, the constructor can be overloaded with multiple types; and the *move constructor* has been introduced, as we can see in Figure 1. The move constructor steals the content of the argument object, and set it to an empty but valid (destroyable) state.

Note that, the actual parameter passed in the place of an overloaded constructor may be either an rvalue or an lvalue. When the object passed as parameter is referred by a name, then it is lvalue, otherwise it may be an rvalue. This rule is explained in Figure 2. This is one of the situations where C++ programmers can easily make mistakes, and no errors or warnings will be generated by the compiler.


```

class Base
{
public:
    Base(const Base& rhs); // copy ctor
    Base(Base&& rhs);      // move ctor

    // ...
};

```

Fig. 1. Copy- and move constructors.

```

Base&& f();

void g(Base&& arg_)
{
    Base var1 = arg_; // copy Base::Base(const Base&)
    Base var2 = f();  // move Base::Base(Base&&)
} // arg_ goes out of scope here

```

Fig. 2. Named rvalue explained.

```

class Derived : public Base
{
    Derived(const Derived& rhs); // copy ctor
    Derived(Derived&& rhs);      // move ctor -> move semantics

    // ...
};

```

Fig. 3. Derived class from Base.

```

// wrong
Derived(Derived&& rhs) : Base(rhs) // wrong: rhs is an lvalue
{
    // Derived-specific stuff
}

// good
Derived(Derived&& rhs) : Base(std::move(rhs)) // good, calls Base(Base&& rhs)
{
    // Derived-specific stuff
}

```

Fig. 4. Wrong and good move constructors.

In Figure 3 we can see a derived class from the Base class. The Derived class has a copy constructor and a move constructor. The implementation of the copy constructor using costly copy operations, while the move constructor uses move semantics.

Because of the rule explained above, if an object has a name, then it behaves like an lvalue. In this case, we must explicitly call the `std::move` function to enforce the move semantics. Without it, the `Base(const Base& rhs)` will be called, i.e. the Base part of the object will be copied instead of using the move semantics. This implies an obvious error in the code – as we can see in Figure 4.

```

template<class T>
void swap(T& a, T& b)
{
    T tmp(std::move(a)); // enforce move semantics
    a = std::move(b);
    b = std::move(tmp);
}

```

Fig. 5. swap with move semantics.

```

template<class T>
void swap(T& a, T& b)
{
    [[move]] T tmp(std::move(a));
    [[move]] a = std::move(b);
    [[move]] b = std::move(tmp);
}

```

Fig. 6. The swap function with annotated statements.

The situation in Figure 5 is similar. It is critical to write the `std::move` call in the first line, because the variable `a` has a name. Without the `std::move` it will be copied to `tmp`, and the code will likely be executed slower than as assumed. Our prototype tool is analyzing such issues to detect the unnecessary copy operations.

3. ATTRIBUTE-BASED ANNOTATIONS AND CHECKING

In C++11 a new feature has been introduced to able the programmers to annotate the code, and to support more sophisticated domain-specific language integration without modifying the C++ compiler [Porkoláb and Sinkovics 2011; Sinkovics and Porkoláb 2012]. The new feature is called *generalized attributes* [Kolpackov 2012; J. Maurer 2008]. Currently this is rarely used, because the lack of standard custom attributes, but it is a great extension opportunity in the language.

Most important C++ compilers, like GNU g++ and the Clang compiler (which is a C++ frontend for the LLVM [Klimek 2013; Lattner et al. 2014]) parses the generalized attributes, binds to the proper Abstract Syntax Tree (AST) node even if Clang displays a warning, because all generalized attributes are ignored for code generation. Even if the attributes are ignored for code generation, they are included in the abstract syntax tree, and they can be used for extension purposes. In our case, we will annotate functions and statements about the expected copy/move semantics. For example, the original code in Figure 5 can be annotated with the `[[move]]` attribute as can be seen in Figure 6.

Though, the statements are validated correctly, the annotations are redundant and every line starts with the same `[[move]]` attribute. To avoid this, the whole function can be annotated as can be seen in Figure 7. Annotating the whole function changes the default semantics of the statements inside the function – the unannotated functions have copy semantics by default.

Our validator tool use two different annotations: the `[[move]]`, and the `[[copy]]`. To determine the expected semantics of a statement needed the default behavior of the function (which defaults to `[[copy]]` in the most cases, but it can be overridden), and the optional statement annotation. If the statement does not have annotations, then the semantics of the statement is the same as the semantics of the function. If the statement contains e.g. a constructor call, then the tool checks whether the proper constructor call is made. When not the proper constructor is called, then the tool displays an error message.

```

template<class T>
[[move]]
void swap(T& a, T& b)
{
    T tmp(std::move(a));
    a = std::move(b);
    b = std::move(tmp);
}

```

Fig. 7. The annotated version of the swap function.

Furthermore, our tool validates the implementation of the move constructors. As can be seen in Figure 4, easy to make mistakes in the move constructor. As to prevent these errors, the tool sets the default semantics of the move constructors to `[[move]]` instead of `[[copy]]`.

Note that, the builtin types, like `int` and `long` does not have constructors, so the tool will allow to copy primitive types even the `[[move]]` is set.

4. FUTURE WORK

The functionality of the tool can be integrated into the compiler itself, and the 3rd party tool problem can be solved. The compiler traverses the whole AST during the compilation (at least when the parser builds the AST), and our tool traverses the AST too. There is a trivial solution for the issue, because the AST visitor of the compiler may do more work at the same time.

Furthermore, we are investigating the possibilities of the comparison-analysis at language level. The first approach is to measure the depth of nested loops. The second step is to identify definite loops – it is hard to detect loops with "read-only" loop variable (the variable must be incremented/decremented in the *step* part of the loop). Note that, these problems appear in code validation process too, because the understandable loops are important in programming.

All of the features in this paper will be implemented in our experimental programming language, called Welltype. This language is an imperative programming language with strict syntax and strong static type-system. This language supports generalized attributes as well, and the attributes can be used by the programmer – the attributes are read-only at runtime, and the dynamic program loader checks their values at link time, whether are matching with the import- and export specification.

5. CONCLUSION

In this paper we presented the problems in real-time and embedded programming, such as high-performance computing and responsive server programs. For such applications giving an exact execution time estimation is very problematic. Many external factors influences the execution: including internal mechanisms in the processor and in the operating system. Instead of investigating absolute time properties, in our research we focused on measuring copy operations – an operation which highly influences C++ execution times.

We shortly described the C++ move semantics, and their positive effects on the programs. The C++ move semantic is a relatively new language level enhancement for optimizing the program execution avoiding unnecessary copy operations, but keep these optimizations at a safe level. Unfortunately, it is easy to make mistakes when working with move semantics. Many operations intended to use move semantics, in fact, apply expensive copy operations instead. Our method targets such mistakes, making operations planned for move semantic marked explicitly by C++11 annotations and checking their implementations for unwanted copy actions.

We implemented a Clang-based prototype tool to detect copy/move semantic errors in C++ programs. Firstly, our tool helps to avoid the negative effects of the unnecessary copies in execution time. Secondly, the validation of the program: it detects the unfortunate cases when the programmer fails to implement a proper move constructor.

Our tool can deal with both regular functions and constructors. The execution time of our tool is linearly depends only the size of the source code.

We recognised that, the functionality of our tool can be integrated into the compiler, and the overhead of detecting copy/move semantics errors can be greatly decreased. Furthermore, we are intended to implement the same functionality in our experimental programming language, called Welltype.

REFERENCES

The C++ Programming Language, 4th Edition.

- A. Alexandrescu. 2001. *Modern C++ Design: Generic Programming and Design Patterns Applied*. Addison-Wesley, Boston, MA.
- Krzysztof Czarnecki and Ulrich W. Eisenecker. 2000. *Generative Programming: Methods, Tools, and Applications*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA.
- A. Gurtovoy D. Abrahams. 2004. *C++ template metaprogramming, Concepts, Tools, and Techniques from Boost and Beyond*. Addison-Wesley, Boston, MA.
- Bach Khoa Huynh, Lei Ju, and Abhik Roychoudhury. 2011. Scope-aware data cache analysis for WCET estimation. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2011 17th IEEE*. IEEE, 203–212.
- M. Wong J. Maurer. 2008. Towards support for attributes in C++ (Revision 6). Open Standards, N2761. (2008). Retrieved Aug 7, 2014 from www.open-std.org/jtc1/sc22/wg21/docs/papers/2008/n2761.pdf
- M. Klimek. 2013. The Clang AST – a Tutorial. LLVM Developers’ Meeting. (April 2013). Retrieved July 28, 2014 from <http://llvm.org/devmtg/2013-04/klimek-slides.pdf>
- B. Kolpackov. 2012. The Clang AST – a Tutorial. (April 2012). Retrieved July 15, 2014 from <http://www.codesynthesis.com/~boris/blog/2012/04/18/cxx11-generalized-attributes/>
- C. Lattner and others. 2014. Clang: a C language family frontend for LLVM. (2014). Retrieved Aug 7, 2014 from <http://clang.llvm.org/>
- Zoltán Porkoláb and Ábel Sinkovics. 2011. Domain-specific language integration with compile-time parser generator library. *ACM SIGPLAN Notices* 46, 2 (2011), 137–146.
- Á Sinkovics and Zoltán Porkoláb. 2012. Domain-specific language integration with c++ template metaprogramming. *Formal and Practical Aspects of Domain-Specific Languages: Recent Developments* 32 (2012).
- ISO International Standard. 2011. ISO/IEC 14882:2011(E) Programming Language C++. (2011).
- Bjarne Stroustrup. 1994. *Design and Evolution of C++*. Addison-Wesley.
- Todd Veldhuizen. 1995a. Expression templates. *C++ Report* 7, 5 (1995), 26–31.
- Todd Veldhuizen. 1995b. Using C++ Template Metaprograms. *C++ Report* 7, 4 (1995), 36–43.
- Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, and others. 2008. The worst-case execution-time problem: overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems (TECS)* 7, 3 (2008), 36.

Clone Wars

VIKTÓRIA FÖRDÖS, ELTE-Soft Nonprofit Ltd.
MELINDA TÓTH and TAMÁS KOZSIK, Eötvös Loránd University

Code clones are unwanted phenomena in legacy code that make software maintenance and development hard. Detecting these clones manually is almost impossible, therefore several code analyser tools have been developed to identify them. Most of these detectors apply a general token or syntax based solution, and do not use domain specific knowledge about the language or the software. Therefore the result of such detectors contains irrelevant clones as well. In this paper we show an algorithm to refine the result of existing clone detectors with user defined domain specific predicates to preserve only useful group of clones and to remove clones that are insignificant from the point of view defined by the user.

Categories and Subject Descriptors: D.2.3 [Software Engineering] Coding Tools and Techniques; D.2.7 [Software Engineering] Distribution, Maintenance, and Enhancement; F.2.2 [Analysis of algorithms and problem complexity] Nonnumerical Algorithms and Problems

General Terms: Languages, Design

Additional Key Words and Phrases: Grouping, Filtering, Clone detection, Erlang, Suffix tree, Static program analysis

1. INTRODUCTION

Code clones, the result of the "copy&paste" programming technique, have negative impact on software quality and on the efficiency of the software maintenance process. Although copying may be the fastest way of creating a new feature, after a while it is really hard to detect and maintain the multiple instances of the same code snippets.

Based on static source code analysis, clone detectors try to identify code clones automatically. Several clone detectors exist [Roy et al. 2009] applying different techniques to select the clones. These techniques include string, token, syntax and also semantics based approaches.

In the context of the Erlang programming language [Armstrong 2007], there are three clone detectors [Li and Thompson 2009; Fördös and Tóth 2014b; 2013] implementing different techniques to select duplicated code. Although the clones identified by these techniques can be considered duplicates, some of them are *irrelevant* in certain points of view. The filtering system of Clone IdentifiErl allows users to tailor the result in different ways using domain specific knowledge about the language.

This filtering technique can be easily applied on duplicate code detectors that yield clone pairs [Fördös and Tóth 2014b; 2013]: it simply leaves out the pairs which do not fulfil the requirements. When a clone detector groups the identified clones [Baker 1996; Koschke 2012; Fördös and Tóth 2014a], the result is more comprehensible, but makes the filtering less straightforward. Filtering out some part of a group of clones results in smaller groups of clones. Sometimes smaller means that we have less group members, in other cases we have smaller clones – or both.

This work is supported by Ericsson–ELTE-Soft–ELTE Software Technology Lab.

Author's address: V. Fördös, 1/C Pázmány Péter sétány, 1117 Budapest, Hungary; email: f-viktoria@elte.hu; M. Tóth, 1/C Pázmány Péter sétány, 1117 Budapest, Hungary; email: tothmelinda@elte.hu; T. Kozsik, 1/C Pázmány Péter sétány, 1117 Budapest, Hungary; email: kto@elte.hu.

Copyright © by the paper's authors. Copying permitted only for private and academic purposes.

In: Z. Budimac, T. Galinac Grbac (eds.): Proceedings of the 3rd Workshop on Software Quality, Analysis, Monitoring, Improvement, and Applications (SQAMIA), Lovran, Croatia, 19.-22.9.2014, published at <http://ceur-ws.org>.

In this paper we show a general, language independent algorithm to refine the result of existing clone detectors that produce groups of clones. We apply domain specific predicates to the clones to filter out useful groups of clones from different points of view. For example, the clone elimination process can be simplified by removing clone instances that are difficult to be eliminated. The filtering system can be also used to exclude those clones from the result that are duplicates of any given exceptions. For instance, consider that the generated source code should be removed from the results. We also emphasize here that maintenance time can be decreased by focusing on the clones that cause the hardest problems, so the developer can work on the most useful maintenance tasks.

2. RELATED WORK

Clone detection is a wide field of research. Here we focus on filtering and grouping techniques.

Several detectors for duplicates exist, but only a few of them concentrate on functional languages, such as [Brown and Thompson 2010] developed for Haskell, and Wrangler [Li and Thompson 2009] for Erlang. We have proposed Clone IdentifiErl [Fördős and Tóth 2013] for Erlang, which is an AST/metric based approach. We have also published a purely metric driven algorithm [Fördős and Tóth 2014b] that characterises the Erlang language by using software metrics to identify clones in Erlang programs. In Clone IdentifiErl, a new standalone extensible filtering system has been introduced to filter out irrelevant clones, whilst in [Fördős and Tóth 2014b] we have given a filtering system that is capable of removing both irrelevant and false positive clones. The papers [Fördős and Tóth 2014b; 2013] argued for the necessity of this step, and presented a domain-specific implementation for Erlang.

Earlier, Juergens and Göde [2010] have proposed an iterative, configurable clone detector (ConQAT) containing a filtering system. ConQAT can remove repetitive generated code fragments and overlapping clones by iteratively reconfiguring and rerunning its initial clone detector.

Different clone detection techniques have been used by known detectors. Some of them, e.g. the suffix-tree algorithm [Baker 1996], form groups from the resulting clones. On the other hand, there are algorithms that produce clone pairs. In this latter case, it is also possible to group the results, but this has an additional computational overhead. For example, paper [Fördős and Tóth 2014a] shows a general and broadly usable method to group the result of clone detection algorithms.

Although significant research has been carried out in this area, grouping and filtering in one step is a novel technique. The method, presented in this paper, does not conflict with the already existing techniques and tools: it is an additional tool to refine existing results.

3. IMPROVING CLONE DETECTION

There are many algorithms for detecting code duplicates. They differ in accuracy as well as in execution time. Our goal is to improve accuracy without compromising efficiency. In this paper, we propose a standalone, and also a language independent, approach that can be used after any duplicated code detector to tailor its results. Here, we also present how it can be configured to facilitate accurate clone detection in Erlang programs.

Clone detectors result in either clone pairs or clone groups. Regardless of the format of the result, our algorithm can improve it until pairs can be considered as groups of two-elements. Therefore, our algorithm is defined to work with *initial clone groups*. It examines each initial clone group whether all elements of the group are “relevant” enough. The goodness of an element is judged by easily replaceable *filters*, thus the behaviour of the algorithm can be customised to fit various purposes. Our algorithm decomposes a group into sub-groups on which all the filters hold, and removes irrelevant elements from the result.

The algorithm provided here assumes certain properties for initial groups. Namely, each clone in a group must have the same amount of building blocks. In our Erlang implementation, the building

block is a *top-level expression*, where a top-level expression refers to the main expression or to a sequence of main expressions making up a function clause. There are many clone detectors [Roy et al. 2009] that produce initial clone groups for which the required property holds. Note that a well-known detection technique, the suffix tree based algorithms [Baker 1996; Koschke 2012], provide initial clone groups with the required property. A clone detector employing suffix tree [Gusfield 1997] has been implemented in Wrangler [Li and Thompson 2009] (and also in RefactorErl [RefactorErl project 2014]), and we will use this kind of initial groups for the practical evaluation of our approach in this paper. Before we detail our algorithm, we briefly review the general clone detection technique that uses suffix tree.

3.1 Suffix tree

Usually, clone detectors that use suffix tree are token-based algorithms. Tokens of the analyzed program are mapped to words of a formal language based on the token kind. For instance, identifiers and literals can be mapped to ‘a’, the begin keyword is mapped to ‘b’, the case keyword is mapped to ‘c’ and so on. The suffix tree is constructed from using the transformed tokens. The groups of initial clones are gathered as subtrees from the entire suffix tree. This step requires $\mathcal{O}(n * \log n)$ (n denotes the number of tokens) steps in the worst case.

The main advantages of this technique are the low computational cost and the compact result, because it demands no further grouping. Several duplicated code detectors [Baker 1996; Koschke 2012] use this algorithm as their initial clone detectors, because suffix tree based clone detection is a general technique that can be easily applied to detect clones in any programming language.

However, its general applicability implies its weak points. It is a token-based detector with no built-in knowledge of programming languages, thus several clones forming no valid syntactical unit may appear in its result. These clones need to be further cut to meet the syntactical rules of the programming language in which the clones were implemented. Hereupon, it produces several useless clones that consist of a few tokens and have nearly no syntactic characteristics. Last but not least, finding gapped clones becomes impossible, because the algorithm of suffix tree cannot deal with these clones.

3.2 Filtering

Our algorithm examines each initial clone group whether its elements satisfy the required properties described by the user defined filters. Filters are applied to the building blocks of the clone instances that belong to the same group. Filters can be chosen arbitrarily to fit any purpose.

Our algorithm allows developers to concentrate only on important clones by removing irrelevant clones from the result, as Clone IdentifiErl does. Our algorithm can also come to rescue, if the result needs to be cleaned by excluding elements that are required to be duplicated. For instance, consider constraints originating from business logic, or the cases when a function acts as a bridge that connects two applications. Expressions referring to these functions are obviously clones, but they are necessarily present. For another example, consider that the examined source code contains a parser (e.g.: `yacc`) generated source code which can be excluded from the clone groups by using our algorithm. To best of our knowledge, no Erlang specific approach can handle such exceptions.

In this paper, we show how our algorithm can ease the elimination process by using Erlang specific filters. Although a suffix tree based detector is being employed to produce the initial clones, our implementation already ensures that the initial clones of a group are real syntactic clones that differ only in the used identifiers and operators. Thus, we only want to check that the following two predicates hold for a group to ease the elimination process.

—The elements of the group refer to nearly the same set of functions.

```

% First instance                                % Second instance
?Query:exec(Nd, ?Query:seq([?Expr:clause(),      ?Query:exec(RecNode, ?Query:seq([?Rec:file(),
                                     ?Clause:form(),          ?File:included(),
                                     ?Form:func()])))          ?File:module()])))

```

Fig. 1. A clone whose instances refer to different functions

$$\begin{array}{c}
 \begin{array}{ccccc}
 & 1 & 2 & 3 & 4 & 5 \\
 \begin{array}{c} 1 \\ 2 \\ 3 \end{array} & \left(\begin{array}{ccccc}
 a & x & a & y & a \\
 b & c & b & c & b \\
 d & e & d & e & z
 \end{array} \right)
 \end{array}$$

Fig. 2. Record types referred by expressions in five three-unit long clone instances of a group

$$\begin{array}{c}
 \begin{array}{ccccc}
 & 1 & 2 & 3 & 4 & 5 \\
 \begin{array}{c} 1 \\ 2 \\ 3 \end{array} & \left(\begin{array}{ccccc}
 \textcircled{a} & x & \textcircled{a} & y & \textcircled{a} \\
 \textcircled{b} & \textcircled{c} & \textcircled{b} & \textcircled{c} & \textcircled{b} \\
 \textcircled{d} & \textcircled{e} & \textcircled{d} & \textcircled{e} & z
 \end{array} \right)
 \end{array}$$

Fig. 3. Maximal sized groups containing one-unit long clone instances

—The elements of the group refer to the same record definitions.

The first property excludes clones whose elements call mostly different functions, because the functionalities implemented in these clones are likely to be independent. Thus, the elimination of these clones is not a high-priority task. For instance, consider Figure 1. The latter property is greatly Erlang specific. Note that these filters should only be replaced when tailoring the algorithm to report clones written in another programming language.

3.3 Grouping & filtering in one step

In this section we briefly explain how to process an initial clone group by filtering and regrouping clones. The algorithm decomposes a group containing m pieces of n -unit long clone instances into subgroups based on filtering results.

The original group can be best understood as an expression matrix of size $n \times m$: a column in this matrix is a clone instance, i.e. a sequence of n (top-level) expressions appearing in the program that was categorized by the initial grouping as a clone of the other columns in the same matrix. A row in the matrix contains m occurrences of a “similar” expression. For efficiency of the initial clone detection phase, this similarity may be too permissive. We can design more specific “filters” to express domain-specific knowledge about “relevant” clones, by considering two expressions similar in a more restrictive manner. For instance, we can introduce a filter which considers two expressions different if they refer to different record types – even if they were declared similar by the initial clone detection.

A subgroup is formed as an intersection of some selected rows and columns from the original expression matrix. Columns of a subgroup are clones that are relevant from the point of view of the applied filtering mechanism. Our algorithm will try to find maximal sized sub-groups, with elements consisting of as many units as possible.

For the sake of the example, assume that we characterise expressions based on the referred record types. Let us denote an expression with ‘a’ if it refers to a record type ‘a’. In Figure 2, a characterisation of an initial clone group can be seen. The group contains five (i.e.: $m = 5$) three-unit long clone instances (i.e.: $n = 3$). The elements of the matrix represent the records referred by the expressions of the clones; the third element of the first row is an expression referring to record ‘a’ (only). Note that only two of the initial clones in the group are considered relevant by this filtering: column 1 and column 3. Furthermore, our algorithm will identify a shorter clone consisting of two top-level expressions c, e in columns 2 and 4 as well.

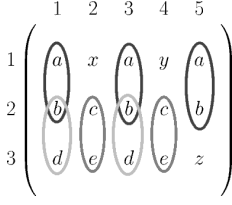


Fig. 4. Joining sub-groups

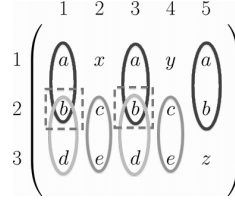


Fig. 5. Finding sub-groups to be glued

We want to maximise both the length (number of columns) and the size (number of rows) of every constructed sub-group. These properties are orthogonal to each other, therefore the maximisations of them impede each other.

We propose an iterative algorithm. We start by identifying sub-groups having one-unit long clone instances. For our example, such sub-groups are shown in Figure 3: in each row, the elements that belong to the same sub-group are painted using the same shade of grey. Obviously, we create maximal sized sub-groups. In the third row, for instance, we could identify two sub-groups (one for d and another one for e), both containing two columns. In the first row, however, we have a sub-group with three columns.

Next, we try to improve the other dimension, i.e. to lengthen the elements of sub-groups. This goal is achieved in two steps. First, we try to join the previously determined sub-groups; here care must be taken not to lose any existing maximal sized sub-groups. We can join two sub-groups if there are no rows between them (clones are continuous blocks of top-level expressions), and they share at least two columns (i.e. at least two clones contain the matching expressions). Furthermore, if there is a clone instance in any of the to-be-joined sub-groups that is not included in the newly created sub-group, then the original sub-group containing this clone instance must be preserved (otherwise it can be thrown away). We will come back to this *covering* problem soon.

Joining is illustrated in Figure 4. Note that matrix elements may belong to multiple sub-groups, as happened with the b s in the second row of the expression matrix. We could join those b s with the a s of the first row, as well as with the d s of the third one.

In the second step of sub-group lengthening, we iteratively glue overlapping sub-groups together. Check the left half of Figure 5: two groups ($a;b$ in the first two rows and $b;d$ in the last two rows) are glued together based on the overlapping in the second row (dashed rectangles). Naturally, it is required that glued sub-groups have at least two common columns: without that they would not be clones. Here the sub-groups share the first and the third columns.

Again, if a clone instance that belongs to any of the input sub-groups is not present in the glued sub-group, then its containing sub-group must be preserved. (We will refer to this phenomena as the new sub-group *not covering* the old one.) This can be observed in the right half of Figure 5. After constructing the new sub-group $a;b;d$ in columns 1 and 3, we can drop sub-group $b;d$, but we must preserve the sub-group $a;b$ in the first two rows, since it contains the third column, which is not included in the new sub-group, and hence the new sub-group does not cover the original $a;b$ sub-group.

The gluing step is repeated until there are no sub-groups that can be glued together. Then the algorithm terminates, and outputs the determined sub-groups as a refined grouping of clones.

4. FORMAL DESCRIPTION

Now we define our filtering&grouping algorithm more precisely. The algorithm operates on a single clone group, represented as an expression matrix of size $n \times m$. Each column represents a sequence of top-level expressions in a function clause (a clone instance), and a row corresponds to similar expres-

sions with respect to an initial clone detection algorithm.

$$G \in \mathcal{E}^{n \times m}$$

For filtering out irrelevant clones, we will use a reflexive and symmetric (but not necessarily transitive) binary relation over expressions.

$$f \subseteq \mathcal{E} \times \mathcal{E}$$

Our algorithm takes a clone group G and a filtering relation f , and produces a set of subgroups, which refines the grouping G .

$$G, f \mapsto \{G_1, G_2, \dots, G_r\}$$

Each subgroup G_i represents a clone group made up of m_i clone instances, each instance having length n_i , where $1 \leq n_i \leq n$ and $2 \leq m_i \leq m$.

$$G_i \in \mathcal{E}^{n_i \times m_i}$$

A subgroup selects a block of rows and some of the columns of the original expression matrix. The initial clones represented by the selected columns in the subgroup contain an expression sequence (the selected rows) which is accepted as a “relevant” clone.

We can represent a subgroup with a “selection” s , relative to an initial group G . The block of rows selected by s is denoted by $s.r$, where $s.r.\ell$ is the lower, and $s.r.u$ is the upper bound of the selection. The set of columns selected by s is denoted by $s.c$. (To improve the efficiency of the algorithm, $s.c$ can be represented as an ordered list of numbers.)

$$s = (r, c) \text{ where } r = (\ell, u), \quad \ell \in [1..n], \quad u \in [\ell..n], \quad c \subseteq \{1, \dots, m\}$$

The algorithm is defined as two steps (Sections 4.1 and 4.2, respectively) followed by an iteration of a third step (Section 4.3). No more than $n - 2$ iterations of the third step are needed; the algorithm can terminate earlier if fixed point is reached.

4.1 One unit long clone instances

The first step of the algorithm produces S_1 , a set of selections of the initial group.

$$G, f \mapsto S_1$$

Each clone instance of each selection in S_1 has length 1 (these clone instances are formed from only one expression). This is the only step of the algorithm where filtering takes place, and predicate f is used. All pairs formed from the elements of each selection in S_1 satisfy predicate f . In the subsequent steps we shall maximize both the length of reported clone instances, and the size of the subgroups.

$$S_1 = \bigcup_{i=1}^n \left\{ ((i, i), c) \mid c \in \text{MaxProperCliques}(\text{graph}(f, G, i)) \right\}$$

where $\text{graph}(f, G, i)$ is the graph of f regarding to the expressions of the i^{th} row in G with vertices $\{1, \dots, m\}$ and edges

$$\left\{ (p, q) \mid (G[i, p], G[i, q]) \in f \right\}$$

and $V \in \text{MaxProperCliques}(g)$ means that V is a clique (the vertices of a complete subgraph) of g which contains at least 2 vertices, and V is not included in a larger clique (i.e. V is inclusion-maximal [Bomze et al. 1999]).

4.2 Joining clone instances

The second step of the algorithm takes clone subgroups containing one unit long clones, and try to join subgroups.

$$S_1 \mapsto S_2$$

Joining can be defined in two steps. First, we introduce S'_1 as follows.

$$S'_1 = \left\{ ((\ell_1, u_2), c_1 \cap c_2) \mid ((\ell_1, u_1), c_1) \in S_1, ((u_1 + 1, u_2), c_2) \in S_1, |c_1 \cap c_2| > 1 \right\}$$

Let the binary relation covers over selections be defined as a partial order in the following way: s_1 covers s_2 if and only if

$$s_2.c \subseteq s_1.c \wedge s_1.r.\ell \leq s_2.r.\ell \wedge s_2.r.u \leq s_1.r.u$$

Finally, we can provide S_2 by combining S_1 and S_2 and eliminating selections that are already covered by other, larger selections.

$$S_2 = S'_1 \cup S_1 \setminus \{s \mid \exists s' \in S'_1 : s' \text{ covers } s\}$$

4.3 Glueing clone instances

The third step, which must be repeated until fixed point is reached (which will happen after no more than $n - 2$ iterations) is also described in two steps.

$$S'_i = \left\{ s \mid s_1, s_2 \in S_i, s_1.r \text{ overlaps with } s_2.r, s.r.\ell = \min(s_1.r.\ell, s_2.r.\ell), s.r.u = \max(s_1.r.u, s_2.r.u), \right. \\ \left. s.c = s_1.c \cap s_2.c, |s.c| > 1 \right\}$$

where two blocks of rows are overlapping, i.e.

$$(\ell_1, u_1) \text{ overlaps with } (\ell_2, u_2) \text{ if and only if } (\ell_1 \leq \ell_2 \leq u_1) \vee (\ell_2 \leq \ell_1 \leq u_2).$$

Now we can define S_{i+1} by removing all the selections from S'_i that are covered by other, larger selections.

$$S_{i+1} = S'_i \setminus \{s \mid \exists s' \in S'_i : s' \text{ covers } s\}$$

When the iteration of this third step reaches fixed point, the last set of selections, S_t can be used to determine the set of subgroups returned by our algorithm. For each selection $((\ell, u), c) \in S_t$, we yield a subgroup of size $(u - \ell + 1) \times |c|$, containing the intersection of the selected rows and columns of the initial expression matrix.

5. CONCLUSIONS

In this paper, we proposed a broadly usable filtering algorithm that quickly removes those clones from the results that are insignificant from the point of view defined by the user. The proposed algorithm is language independent, thus the results of many duplicated code detectors can efficiently be improved. By removing irrelevant clones, the maintenance costs can be decreased, because the programmers need to only deal with important issues.

In this paper, we defined rules that are specialised for easing the clone elimination process in Erlang programs. We discussed the underlying ideas, and we also gave a formal description of our algorithm.

We note that we successfully evaluated the realisation¹ of the algorithm and assessed the results. All of our goals were reached; clones that are hard to eliminate are not present in the results. The filtering

¹The authors would like to thank to Bence Szabó for the implementation.

phase requires only a small extra computational cost that is infinitesimal. Moreover, it removes clones that are insignificant from the point of view defined by the user. Thus, the algorithm quickly cleans the result and helps programmers focus on only important cases.

Future work will consist of evaluating the proposed algorithm by using initial clones reported by different clone detectors and studying and comparing the results of these test runs. Differences in the results will indicate areas for future study.

REFERENCES

- Joe Armstrong. 2007. *Programming Erlang: Software for a Concurrent World*. Pragmatic Bookshelf.
- Brenda S. Baker. 1996. Parameterized Pattern Matching: Algorithms and Applications. *J. Comput. System Sci.* 52, 1 (1996), 28–42. <http://www.sciencedirect.com/science/article/pii/S0022000096900033>
- Immanuel M. Bomze, Marco Budinich, Panos M. Pardalos, and Marcello Pelillo. 1999. The maximum clique problem. *Handbook of Combinatorial Optimization (Supplement Volume A)* 4 (1999), 1–74. <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.56.6221>
- Christopher Brown and Simon Thompson. 2010. Clone Detection and Elimination for Haskell. In *PEPM'10: Proceedings of the 2010 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation*, John Gallagher and Janis Voigtlander (Eds.). ACM Press, 111–120. <http://www.cs.kent.ac.uk/pubs/2010/2976>
- Viktória Fördös and Melinda Tóth. 2013. Identifying Code Clones with RefactorErl. In *Proceedings of the 13th Symposium on Programming Languages and Software Tools, ISBN 978-963-306-228-9*. Szeged, Hungary, 31–45.
- Viktória Fördös and Melinda Tóth. 2014a. Comprehensible presentation of clone detection results. In *Proceedings of the 4th Symposium on Computer Languages, Implementations and Tools (accepted)*.
- Viktória Fördös and Melinda Tóth. 2014b. Utilising the software metrics of RefactorErl to identify code clones in Erlang. In *Proceedings of 10th Joint Conference on Mathematics and Computer Science (Informatica)*, Vol. LIX. Studia Universitatis Babeş-Bolyai, Cluj-Napoca, Romania, 103–118. Issue Special Issue 1.
- Dan Gusfield. 1997. *Algorithms on strings, trees, and sequences: computer science and computational biology*. Cambridge University Press, New York, NY, USA.
- Elmar Juergens and Nils Göde. 2010. Achieving Accurate Clone Detection Results. In *Proceedings of the 4th International Workshop on Software Clones (IWSC '10)*. ACM, New York, NY, USA, 1–8. <http://doi.acm.org/10.1145/1808901.1808902>
- R. Koschke. 2012. Large-Scale Inter-System Clone Detection Using Suffix Trees. In *Software Maintenance and Reengineering (CSMR), 2012 16th European Conference*. 309–318.
- Huiqing Li and Simon Thompson. 2009. Clone detection and removal for Erlang/OTP within a refactoring environment. In *Proceedings of the 2009 ACM SIGPLAN workshop on Partial evaluation and program manipulation (PEPM '09)*. ACM, New York, NY, USA, 169–178. <http://doi.acm.org/10.1145/1480945.1480971>
- RefactorErl project. 2014. Suffix tree based duplicate code analysis. (July 2014). <http://pnyf.inf.elte.hu/trac/refactorerl/wiki/SuffixTreeBasedDuplicateCodeAnalysis>
- Chanchal K. Roy, James R. Cordy, and Rainer Koschke. 2009. Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach. *Sci. Comput. Program.* 74, 7 (May 2009), 470–495. <http://dx.doi.org/10.1016/j.scico.2009.02.007>

Information and Information Security

JAAK HENNO, Tallinn University of Technology

Tremendous growth of Information Security issues poses a problem - is most of mankind turning criminals or is something wrong with rules? Are we always trying to secure valuable Information or just data and do the rules correspond to generally accepted behaviour?

In order to understand better issues connected with Information Security we have first establish concise meaning of terms 'Information' and 'Information Security'. Information cannot be considered separately from an Information Processing System (IPS); a message is Information only for some IPS and is used by the IPS for achieving its goals.

In the paper is considered general model of IPSs and their goals. To get some insight of attitudes of users, the milieu of Massively Multiplayer On-line Games (MMOG) is considered. A 'Security Incident' may be just a curiosity; cheating in a game is not considered serious offence by fellow players.

Categories and Subject Descriptors: **H.1.0 [General]:** Information/Entropy—*Information/Processing*; **H.1.1 [Systems and Information Theory]:** Natural Information Processing Systems—*Unified view/methodology*; **H.1.2 [Models and Principles].**

General Terms: Information, Entropy, Information Processing Systems, Information Security

Additional Key Words and Phrases: data, models, games, MMORG

1. INTRODUCTION

Insecurity, threats to information, need to defend information from unauthorized access, use, disclosure, disruption, modification, perusal, inspection, recording or destruction are nearly always deliberate. Accidents, which in 90ties were responsible for most Information Security issues - a plane flows into an office building destroying all computers in it, employee mistakes ("Format C:"), fire, flood, earthquake, lightning, shooting or otherwise destroying a computer in fit of anger ("You do not have right to access folder 'My Documents'!"), issues with ISP or WAN etc account currently only for tiny part of Information Security problems. According to recent report from Panda Labs [PandaSecurity 2014], in 2013 appeared about 82,000 new malware threats per day and in the whole year - 30 million new malware threats, Kaspersky Lab is detecting 315,000 new malicious files every day [Kaspersky 2014] most of them (>70%) - trojans, especially designed for stealing/damaging Information.

Current computerized Information Processing systems are vulnerable, rigid and not adaptive, since the main focus in their development is on computer technology and communication protocols. System environments, culture of system users, reasons for attacks, culture and operating modes of attackers are considered less.

Threats to Information are not specific only to computers and computer networks, they are present in nearly any Information Processing System (IPS) - social and business organizations, governments, all kinds of living systems down to simplest ones - cells and bacteria. In order understand mechanisms which provoke misuse of information and design and create adaptive information security systems, which can adequately respond to constantly changing dynamic environment and threats and can secure functioning of IPS under attacks and threats the IPS should be considered on more general level. But we should also re-consider rules and practices – do they agree with general understanding of right and wrong?

Author's address: J. Henno, Tallinn University of Technology, Ehitajate tee 5, 19086 Tallinn; email: jaakatcc.ttu.ee

Copyright © by the paper's authors. Copying permitted only for private and academic purposes.

In: Z. Budimac, T. Galinac Grbac (eds.): Proceedings of the 3rd Workshop on Software Quality, Analysis, Monitoring, Improvement, and Applications (SQAMIA), Lovran, Croatia, 19.-22.9.2014, published at <http://ceur-ws.org>

2. INFORMATION FLOOD

Information regulates all processes and information processing is the central characteristics of all Universe. Advances of Biology, Economy, Information Technology, Social studies etc have introduced and supported the understanding, that we are living among Information Processing Systems Ko Seth Lloyd from MIT calculated in 2006 [Lloyd 2006], that the visible universe has so far computed about 10^{122} operations on 10^{92} bits.

Even a simple unicellular organism is a complex and purposefully organized algorithm, but the most complex IPS are humans. Man is the most complex information-processing system on Earth. By some estimates [Spiro 1980] the human brain can store 2.5 petabytes (ca million gigabytes); in the human body are in every second processed 3.4×10^{19} bits.

Our individual Information Processing capabilities are tremendously amplified by our Society and our greatest invention - Internet; so much so, that we clearly are not any more able to manage the process. The amount of digital data created in World grows in two years (at least) twice and in 2020 will be 40000 exabytes (exabyte = 1018 bytes) - over 300 times more than it was in 2005 [EMC 2014]. World population in 2020 is estimated to be 7.72 billion, thus the memory capacity of the whole mankind can in 2020 store only 0.00004% of the whole digital data generated in our digital universe.

We believe, that we are rational beings and everything we do has a purpose. What is the purpose of this flood of data ? Some people call this 'Information Age', 'Age of Big Data', but what is the purpose of amounts of data, what we can not manage? Software is eating the world [Andreessen 2011], who will be in control in 2020 - we or our programs? How much of this data is Information, what has for us some use? How much of this should be protected using the Digital Rights Management (DRM) methods and is this protection in interests of whole Mankind?

3. WHAT IS INFORMATION?

We all agree that we now live in "Information Age", we have become an "Information Society" and that information is the main source of value in the global economy. Several scientists have proposed [Stonier 1990, Hefner 1992, Gitt 1994], that that information is a part of the physical universe the same way as matter and energy. Information present in a system is the measure of organization of the system.

But in spite of vast number of papers on Information, Information systems, Information Security etc the meaning of the word "Information" remains abstract and underdefined. The situation has not become essentially better from the famous utterance from father of Cybernetics Norbert Wiener: "Information is information, not matter or energy" [Wiener 1948, p. 132]. In spite of proliferation of information systems, there is still no generally agreed answers to the questions – What is Information? Has Information natural properties and if so, then what are these properties?

The word 'information' is derived from the Latin word 'informare' - "give form to", i.e. information is always represented with some form, structure, pattern. But 'representation' of a concept (e.g. in human mind) is not the concept itself, representations are the result of grounding our sensory perceptions, discovering patterns in perceptions.

The Oxford Dictionary explains 'information' as 'facts provided or learned about something or someone', i.e. information is communicated to receiver.

It is often claimed that Information Theory was created by Claude Shannon, but the main topic of Shannon's research was Communication. Shannon always considered communication and did not speak about information; he always used notions 'communication, communications channel'. In his ground-breaking paper "A Mathematical Theory of Communication" [Shannon 1948] he explicitly states: "fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point", but "semantic aspects of communication are irrelevant to the engineering problem". Shannon considered messages and communication without any assumptions about their meaning, but information is based on meaning. A message without meaning does not contain information for receiver.

Shannon's paper started the 'Information Theory' boom in many fields- linguistics, biology, physics, so that he was forced to warn: "Workers in other fields should realize that the basic results of the subject [communication channels] are aimed in a very specific direction, a direction that is not necessarily relevant to such fields as psychology, economics, and other social sciences".

Meaning of communicated signals, how these signals are used in receiving IPS both in mechanical and biological systems, i.e. how signals influence, control these systems this was considered by Norbert Wiener considered in his ground-breaking opus "Cybernetics or Control and Communication in the Animal and the Machine" [Wiener 1948]. Wiener considered signals as information. Signals, data become for receiver information if they make sense for receiver and receiver can use them for achieving its goals. A data item is Information for an IPS if it helps the IPS in achieving its goals, i.e if it changes the future behaviour of the IPS, acts for the IPS like a program.

4. INFORMATION PROCESSING SYSTEMS

All Information Processing systems (IPS) - living systems, businesses, social organizations, governments, languages, computer programs, etc are finite and have goals, their purpose is to perform some actions aimed to fulfil their goals and they survive by constantly processing information about threats and opportunities in the world around them.

Goals of living systems are metabolism (getting energy needed for their functioning), growth and reproduction, securing resources/territory and collaboration, goals of businesses - acquiring resources (money) by production of some goods and/or services, growth and creating daughter businesses, goals of languages - improve communication of language users, goals of governments - guarantee and improve well-being of citizens etc. Actions of IPS systems are based on information they receive from their environment; these actions allow them survive and develop, become more complex, reduce their inner entropy. Thus all IPS have (at least) these goals:

1. *metabolism - acquiring energy needed of its actions*
2. *growth/reproduction*
3. *securing their existance, resources/territory from external threats*
4. *collaboration with other similar IPS - when this helps in achieveing their goals.*

Problems with Security arise from the third goal.

4.1 Formal Level

Actions for achieving of goals can be performed only if some conditions are true. Conditions can be expressed with predicates, thus we can consider goals as logical predicates.

Signals, data becomes information for an IPS only when they can be used for proving true their goals. A goal for a living system can be e.g. an eatable object:

$$a \wedge eatable(a)$$

For processing (proving a goal predicate true) data should be stored by IPS. Since all IPS are finite, they tend not to store useless data or if something is stored, but could not be used for a long time, then it is forgotten. According to Landauer's erasure principle [Landauer 1961] forgetting, erasing memory always increases thermodynamic entropy in the environment [Plenio 2001].

If the goal is predicate *vitamins* and IPS has already some information, e.g.

$$vitamins \Leftrightarrow \exists x(vitamins(x))$$

$$vitamins(x) \Leftrightarrow (apple(x) \vee pear(x)) \neg chips(x)$$

and IPS gets from its environment grounded information

$$a \wedge apple(a)$$

then it can deduce the goal *vitamins(a)* and perform the action - *eat(a)*.

Shannon 's formula

$$H(x) = -K \sum_i p(x_i) \ln(p(x_i))$$

for calculating entropy of the used above function

$$f(x, y, z) = (x \vee y) \neg z$$

gives

$$H(f) = -\left(\frac{3}{8} \ln \frac{3}{8} + \frac{5}{8} \ln \frac{5}{8}\right) = 0.954$$

When IPS performs the above deduction and thereafter clears the used memory (its finite!), its entropy increases by this amount.

5. EXAMPLE TO CONSIDER - MMOGS

Problems with Security for environment arise from the third goal of IPS - securing their existence, resources/territory from external threats. IPS should keep balance with their environment for sustainable existence, but humans are often driven by greed and often overact, trying to secure more than inevitable, cheat and commit fraud. Every Security incident means, that somebody is violating the accepted rules and practices. The tremendous growth of security accidents poses a problem: wheather the whole Mankind is becoming criminals or is something wrong with rules, the rules do not agree with what is considered normal.

Reasons of our behaviour lay deep in human culture, history and psychology and are difficult to analyse, but for illustrative 'sandbox example' could be used computer games, especially Massively Multiplayer Online Games (MMOG).

The online gaming industry has in the past decades grown rapidly. The virtual economies created within MMOGs often blur the lines between real and virtual worlds so that several companies producing MMOGs have hired an economists to regulate the virtual economy in their games [Plumer 2012].

As a result, the Security problems in MMOGs are becoming increasingly critical. Cheating, virtual frauds, and other security attacks are widespread in the virtual worlds of MMOGs. Gamers have developed illegal methods to obtain virtual wealth and convert it into real wealth in the real world using Real Money Trading, which allows to buy using real money game values (tropies, achievements).

MMOGs are played in massively distributed systems spreaded over many countries with thousands of client processes interweaving on common servers in real time. This greates many security issues and is the reason for the lack of regulations in this industry - it is difficult to impose legal laws in a world that is virtual, "not real", involving several countries.

As a result has appeared a million-dollar business of developing cheating tools, bots for (half) automated gaming, cheats, trainers and walkthroughs and web-sites, where one could get for monthly fee super-human abilities in some game, e.g. see other players through walls [Maiberg 2014].

It is difficult to tell exact percentage of cheating players. In a reacent study involving one on-line racing game [Blackburn et al 2014] was collected information about gamers on the Steam Community global gaming network: Steam has an anti-cheating system which marks cheaters public profile; from more than 12 million analyzed gamers over 700 thousand (ca 6 %) had their profiles flagged as cheaters.

Cheating was earlier considered as a despicable behaviour, but currently the attitude seems to be changing - the study discovered, that "cheaters are well embedded in the social and interaction networks: their network position is largely indistinguishable from that of fair players". And seeing fellow players succeed through cheating creates the idea that it might be worth the risk also to try.

Cheating is often difficult to distinguish from healthy curiosity. Consider a simple on-line Pong game, where player has to move his paddle in order to bounce a ball to target of another player. If player can place its paddle in correct position (ball bounces from paddle), he/she earns some points; if ball misses the paddle, it also bounces back, but player does not earn anything.



Fig. 1. Cosmic Pong - player has to move his paddle to bounce the ball back and earn points; if ball misses paddle, it bounces back from ground and player does not earn anything.

We did such a game in game-programming class. Everything went well until one student started to earn rapidly tremendous scores. It turned out, that he has decreased the y-coordinate of his paddle, so that ball often started to bounce back from the bottom of the paddle creating several bounces, which quickly increased his score.



Fig. 2. Student's modification - paddle's y-coordinate is decreased, so that ball starts bouncing from the bottom of the paddle and creates every time new points.

What should teacher do - prize for ingenuity or punish for cheating, his modification give him clear advantage? After discussion we changed the game: everyone can modify (some) constants, but the modification will be at once distributed to all computers, so that all players are all the time in equal conditions.

REFERENCES

- Andreessen, M. 2011. Why Software Is Eating The World. Brandtæg, P.B. 2013. Big Data, for better or worse: 90% of world's data generated over last two years. Sciencedaily. (2013).
- Burgin, M. 2010. Theory of Information 2010. World Scientific. ISBN-978-981-283-548-2 (2010), 283–548.
- Duncan, T.L. and Semura, J.S. 2007. Information Loss as a Foundational Principle for the Second Law of Thermodynamics. Foundations of Physics. 37, (2007), 1767–1773.
- EMC. EMC Digital Universe Study (2014), <http://www.emc.com/leadership/digital-universe/index.htm>
- Galbraith, J.R. 1973. Designing Complex Organizations. Addison-Wesley.
- Gánti, T. 2003. The Principles of Life. Oxford University Press.
- Gantz, J. and Reinsel, D. 2012. The Digital Universe in 2020. JDC.
- Gitt, W. 1994. Am Anfang war die Information. Am Anfang war die Information.
- Hefner, K. 1992. Evolution of Information Processing Systems. Springer.
- J. Blackburn, A.I. N. Kourtellis, J. Skorvetz, M. Ripenau 2014. Cheating in Online Games: A Social Network Perspective. ACM Transactions on Internet Technology. (2014).
- Jaynes, E.T. 1957. Information Theory and Statistical Mechanics. Phys. Rev. (1957), 620–630.
- KasperskyLab. Number of the year: Kaspersky Lab is detecting 315,000 new malicious files every day. <http://www.kaspersky.com/about/news/virus/2013/number-of-the-year> (2014)
- Landauer, R. 1961. Irreversibility and heat generation in the computing process. IBM Journal of Research and Development. 5, (1961), 183–191.
- Lloyd, S. 2007. Programming the Universe: A Quantum Computer Scientist Takes on the Cosmos. Vintage. (2007), 256.
- Machta, J. 1999. Entropy, information, and computation. Am. J. Phys. 67, 121 (Dec. 1999).
- Maiberg, E. 2014. Hacks! An investigation into aimbot dealers, wallhack users, and the million-dollar business of video game cheating. PCGamer. (2014).
- PandaLab. Annual Report Pandalabs 2013 summary (2014). <http://press.pandasecurity.com/>
- Plenio, M.B. and Vitelli, V. 2001. The physics of forgetting: Landauer's erasure principle and information theory. Contemporary Physics. 42, 1 (2001), 25–60.
- Plumer, B. 2012. The economics of video games. The Washington Post. (2012).
- Reeb, D. and Wolf, M.M. 1962. (Im-)Proving Landauer's Principle. arXiv:1306.4352. (1962).
- Scarrott, G.G. 1986. The Nature of Information. Computer Journal. 32, 3 (1986), 262–266.
- Shannon, C.E. 1993. Collected Papers. Wiley-IEEE press ISBN-. (1993), 1–968.
- Shannon, C.E. 1948. The mathematical theory of Communication. Bell System Technical Journal. 27, 1 (1948), 623–656.
- Shannon, C.E. and Weaver, W. 1949. The Mathematical Theory of Communication. Univ of Illinois Press. (1949).
- Spiro, T.G. and Stigliani, W.M. 1980. Environmental Issues in Chemical Perspective. Suny Press.
- Zuse, K. 1969. Rechnender Raum. Friedrich Vieweg & Sohn.
- Zuse, K. 1967. Rechnender Raum. Elektronische Datenverarbeitung. (1967), 336–344.
- Wiener, N. 1948. Cybernetics or Control and Communication in the Animal and the Machine. MIT Press.
- Williams, M.A. 2008. Representation = Grounded Information. PRICAI. (2008), 473–484.

Formal Specification of Scientific Applications Using Interval Temporal Logic

BOJANA KOTESKA and ANASTAS MISHEV, University SS. Cyril and Methodius, Faculty of Computer Science and Engineering, Skopje
 LJUPCO PEJOV, University SS. Cyril and Methodius, Faculty of Natural Science and Mathematics, Skopje

Scientific applications simulate any natural phenomena in different scientific domains. Moreover, the problems they solve are usually represented by mathematical models. Taking that in advance, these problems can be described by using specific formal notation and mathematical formulas. Scientific applications are usually created by the scientists without using any software development engineering practices. Our main goal is to include formal methods in the testing process of scientific applications. In this paper, we adapt Interval Temporal Logic (ITL) as a flexible notation for describing software applications. We use Tempura framework and Ana Tempura tool for specifying the properties of the scientific software system. The correctness of the code is verified by comparing the results from the program output and functions written in Tempura. This process is especially important when some code changes or optimizations are made. To verify this concept we made a formal description of the code for calculating bound states of the Morse oscillator well.

Categories and Subject Descriptors: D.2.4 [**Software Engineering**]: Software/Program Verification — *Validation, Formal Methods*; G.4 [**Mathematics of Computing**]: Mathematical Software—*Certification and testing, Documentation, Verification*

General Terms: Verification

Additional Key Words and Phrases: Scientific application, Formal Specification, Interval Temporal Logic, Ana Tempura, Morse Oscillator

1. INTRODUCTION

Scientific applications are widely used nowadays in different scientific domains. Numerical simulations performed by scientific applications can solve problems in many research fields such as: computational chemistry, physics, engineering, mathematics, mechanics, informatics, bioinformatics, etc. Scientific application is defined as a software application that simulates activities from the real world by turning the objects into mathematical models [Ziff Davis Publishing Holdings 1995]. Simulations of scientific experiments require powerful supercomputers, high performance computing infrastructures, clusters or Grid computing [Vecchiola et al. 2009].

The testing process of scientific applications is not the same as testing of the commercial software applications. Problems related to testing come from the non formal specification of the scientific applications. Scientists are mostly interested in scientific research achievements and they do not have any documentation or formal specification for their software [Segal 2008]. When some code changes or code optimization should be performed it is usually hard to understand the code. According to Kelly and Sanders [Sanders and Kelly 2008], the risks of developing scientific applications can be divided

Author's address: Bojana Koteska, FCSE, Rugjer Boshkovikj 16, P.O. Box 393 1000, Skopje; email: bojana.koteska@finki.ukim.mk; Ljupco Pejov, FNSM, P.O. Box 1001, 1000 Skopje; email: ljupcop@iunona.pmf.ukim.edu.mk; Anastas Mishev, FCSE, Rugjer Boshkovikj 16, P.O. Box 393 1000, Skopje; email: anastas.mishev@finki.ukim.mk.

Copyright © by the paper's authors. Copying permitted only for private and academic purposes.

In: Z. Budimac, T. Galinac Grbac (eds.): Proceedings of the 3rd Workshop on Software Quality, Analysis, Monitoring, Improvement, and Applications (SQAMIA), Lovran, Croatia, 19.-22.9.2014, published at <http://ceur-ws.org>.

into three categories: complexity of the theory and the difficulty of its validation; implementation risks (code and documentation) and application usage risk (concerning the use of the application by the target user groups). Validation is difficult because the results sometimes must be compared with the results obtained by the physical experiments. Also, scientists usually visually decide whether the results are satisfactory or not. The survey presented in [Koteska and Mishev 2013] shows that most of the interviewed scientists do not prefer formal software engineering testing methods, do not use any testing tools, do not use any specific test case generation technique and some of them do ad-Hoc testing.

To change the current development practices and to improve testing, we propose adapting the method of formal specification and verification of the scientific applications. We chose Interval Temporal Logic (ITL) as a powerful and flexible mathematical notation for propositional and First-order reasoning about periods of time which is used for describing hardware and software systems. We also use the Ana Tempura tool that is built upon C-Tempura and is used for runtime verification of systems. This tool uses ITL and the executable subset Tempura which is an interpreter for executable ITL formulas [Cau et al. 2002]. As a case study, we chose the code for calculating the bound states of the Morse oscillator well which is described in [Bittner 2009].

The rest of the paper is organized as follows. Related work is given in Section 2. In Section 3, we explain the importance of using the formal notation for scientific application description and we present the benefits of using ITL and Tempura. In Section 4, the ITL formal description of the code for calculating the bound states of the Morse oscillator well is given and the results of the testing process are shown. The conclusion and future work are specified in Section 5.

2. RELATED WORK

There are several research papers that present some methods for formal description of software systems and architectures. An overview of how formal (mathematical) methods can be used in the software development cycle and what methods and tools can support software development is given in [Ostroff and Paige 1998]. Siegel and Avrunin [Siegel and Avrunin 2004] write about issues related to finite-state verification techniques when applied to scientific computation software employing the Open MPI (open source High Performance Message Passing Interface implementation). Siegel and Rossi [Siegel and Rossi 2008] applied model checking techniques to BlobFlow (MPI scientific program consisted of 10K lines of code that implements a high order vortex method for solving the two-dimensional Navier-Stokes equations). ITL has been previously used for describing software or hardware systems, state machines or documents. For example, Sciavicco et al. [Sciavicco et al. 2009] consider the problem of formalizing a medical guideline in a logical language. These guidelines are documents supporting the health-care professionals in managing a disease in a patient. Zedan et al. [Zedan et al. 1999] present an object based formal method for the development of real-time systems which is called ATOM. It is based on the refinement calculus and also the formal specification contains ITL description of the behaviour of a real-time system. El-kustaban et al. [El-kustaban et al. 2012] proposed an executable specification model for an abstract transactional memory (lock-free technique that offers a parallel programming model for future chip multiprocessor systems). They used ITL and AnaTempura to build and validate the model. In [Rossi et al. 2004], the authors use temporal logic that combines points, intervals, and dates to formalize the semantics of UML state machines.

The development of scientific applications does not include software engineering practices or formal methods which means that no requirements, formal specification or any kind of documentation could be found. The problems come later when some code modifications or optimization should be performed and nobody knows what the code function is. ITL is usually used for modeling critical software systems, but it really helps when some code changes are made. For example, code changes must be made when some code optimizations are performed which happens very usually while programming scientific ap-

plications because of the memory lack and CPU performance. The program written in Tempura is one way to check the correctness of the results of the program. It also helps for understanding the order of the statements execution at a given moment of time. If no formal program description is used, then the chances for program errors are bigger and the program is less understandable for people being included in the development process later.

The research shows that no formal specification of the scientific applications using ITL/ Tempura are made. Our main goal is to give a formal code description in order to change the current practices, to give a mathematical representation of the scientific problem and to improve the process of verification.

3. FORMAL METHODS IN SCIENTIFIC APPLICATION DEVELOPMENT

This section presents the benefits of making formal description and verification of scientific applications by using Interval Temporal Logic and Tempura.

3.1 The Benefits of Using Formal Methods

Some of the most important benefits of using formal methods are given below [Hall 2005; Groote et al. ; Sommerville 2007; Woodcock et al. 2009; van der Poll 2010; Jaeger 2010; Clarke and Wing 1996]:

- The formal software description is abstract and precise description which means that a human reader can understand the big picture and all ambiguities can be removed.
- Formal description allows users to make rigorous analysis and to determine useful properties such as consistency or deadlock-freedom.
- Using formal methods when developing complex software results in higher quality, more correct software, and discovering errors that may not have been discovered through traditional testing.
- Formal specification allows users to find the problems and ambiguities in the system requirements.
- Formal methods are used for code verification which is attempt to prove the theorem that if certain condition are satisfied the program will achieve the expected results.
- Formal methods facilitate the production of quality and testing. Maintenance phases are shortened.
- The use of formal methods increases the development correctness confidence and has the potential to eliminate some types of errors in the system.
- Formal methods can increase the understanding of the system by removing inconsistencies, ambiguities and incompleteness.

Formal methods are usually applied to the critical systems development, but there are also critical scientific applications, especially in the bioinformatics research field. These applications have critical implications for life sciences and require strong quality assurance [Umarji et al. 2009].

3.2 Interval Temporal Logic and Tempura

Interval Temporal Logic (ITL) is a formalism that is an extension to standard predicate logic which includes time-dependent operators [Moszkowski and Manna 1984]. The key term of ITL is an interval. An interval is defined as a (in)finite sequence of states where each state is a mapping from the set of variables to the set of values [Cau et al. 2002].

Expression in ITL is defined as:

$$exp ::= z \mid a \mid A \mid g(exp_1, \dots, exp_n) \mid \text{ia} : f,$$

where z is an integer value, a is a static variable (its value cannot be changed within an interval), A is a state variable (its value can be changed within an interval), g is a function symbol, f is a formula.

Formula in ITL is defined as:

$f ::= p(exp_1, \dots, exp_n) \mid \neg f \mid f_1 \wedge f_2 \mid \forall v \cdot f \mid \text{skip} \mid f_1; f_2 \mid f^*$,
 where p is a predicate symbol, $;$ is a chop symbol.

Formulas are building inductively as follows:

- Equality: $exp_1 = exp_2$
- Logical connectives: $\neg f$ and $f_1 \wedge f_2$
- Next: $\bigcirc f$
- Always: $\Box f$

The informal semantics can be represented as follows:

- $\imath a$: f - choose value of a such that f holds
- skip - interval with length 1
- $f_1; f_2$ - the interval is decomposed into two intervals (prefix interval such that f_1 holds and suffix interval such that f_2 holds) or the interval is infinite and f_1 holds.
- f^* - the interval is decomposable into a finite number of intervals such that for each of them f holds, or the interval is infinite and it can be decomposed into an infinite number of intervals such that f holds [?].

For example, this is a valid ITL formula: $(I = 2) \wedge \bigcirc(K = 3)$. It can be interpreted as follows: in the current state I is 2 and in the next state K will be 3.

Interval Temporal Logic provides a basis for the programming language Tempura. The main syntactic categories in Tempura are: expressions (can be boolean or arithmetic), statements (temporal formulas that can be simple or compound) and locations (places where values are stored) [Moszkowski 1985]. A formula is executable in tempura if the following three characteristics are satisfied: the formula is deterministic, the length of an interval is known and the values of the variables are known through that interval [De Montfort University 2004].

4. A FORMAL SPECIFICATION OF THE CODE FOR CALCULATING BOUND STATES OF THE MORSE OSCILLATOR WELL

In this section, we refer to the problem of finding the bound states of Morse oscillator (i.e. solving the stationary Schrödinger equation for Morse potential) and we present the formal specification of the code by using ITL and Ana Tempura. The Morse oscillator well have not been modeled yet with ITL and Tempura. The reason that we chose it for modeling is that code is simple to be understand and it also can be considered as an example of a simple scientific application.

4.1 Bound States of the Morse Oscillator Well

Calculation of eigenenergies of bound states of Morse oscillator is a prototypical exercise in quantum mechanics. This is so since the particular potential serves as a model system for studying molecular vibrations. Morse potential has the following form:

$$U(r) = D_e \cdot [1 - \exp(-\beta \cdot (r - r_e))]^2 \quad (1)$$

where D_e denotes the dissociation energy of the corresponding bond, while r_e is the interatomic distance corresponding to the minimum energy of the oscillator. Vibrational Schrödinger equation with the potential of the form (1) is analytically solvable and the corresponding vibrational eigenenergies are given by:

$$E_v = h \cdot c \cdot \left[\left(v + \frac{1}{2} \right) \cdot \omega_e - \left(v + \frac{1}{2} \right)^2 \cdot \omega_e x_e \right] \quad (2)$$

In eq. (2), v is the vibrational quantum number ($v \in 0, 1, 2, \dots$), $\omega_e x_e$ is the so-called anharmonicity constant and is related to the parameter β and the reduced mass of the oscillator μ by:

$$\beta = 2 \cdot \pi \cdot c \cdot \omega_e \cdot \sqrt{\frac{\mu}{2 \cdot D_e}} \quad (3)$$

All other symbols in (2) have their usual meanings. The fact that the vibrational Schrödinger equation for Morse oscillator is analytically solvable makes this system a rather convenient test case for a number of numerical methods aimed to solve the quantum vibrational problem.

In our particular application of the formal code specification approach, we solve the vibrational eigenvalue problem by the discrete variable representation methodology [J. C. Light and J. V. Lill 1985], following closely the approach adopted by Bittner [Bittner 2009]. To solve a problem in quantum mechanics by using numerical methods, the Hamiltonian operator should be represented in a finite polynomial basis. In this case, the Tchebychev polynomials are used as a basis.

One of the methods we made a formal specification for is *thcheby(...)*. It returns a set of points *pts*[NPTS], the eigenstates the Laplacian operator, $(-\partial^2/\partial x^2)$, in the basis *ke_fb*[NPTS] (kinetic energy in finite basis), set of weights *w*[NPTS] and a transformation matrix *T*[NPTS][NPTS]. NPTS is the number of points. There are two representations: finite basis representation (FBR) and discrete variable representation (DVR). The transformation matrix carries one from the FBR to a 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));
}
}
}
```

To solve the bound states of the Morse oscillator well we should define the number of points: NPTS=100, and range: $xmin = -3$ and $xmax = 32$. The goal is to construct the Hamiltonian matrix in the DVR basis and then diagonalize it to determine the eigenvalues and eigenvectors. The eigenvalues (energies) below 0 are bound states. We use the eigenvectors and eigenvalues to plot the wave functions. We automated the part for checking the correctness of wavefunctions by checking the values of the first two eigenvectors. A wave function is correct if the difference between two neighbor values is smaller than 10^{-3} and the values gradually getting tend to zero.

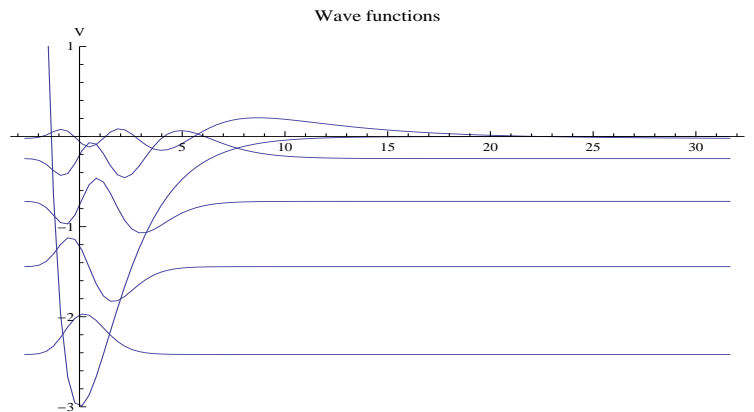


Fig. 1. Wave functions

4.2 The ITL Formal Specification using Ana Tempura

We made a formal specification of the code for calculating the bound states of the Morse oscillator well by writing a Tempura code and making tests to compare the output results from the .exe version of the C program and Tempura code. In order to establish communication between the tempura file and the .exe version of the C program assertions must be added in the C code. We will explain the C and Tempura code where the checking of the values in the **ke_fb[]** array is performed. The code for checking the other arrays and values is similar.

```
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);
}
```

The assertion functions are used for checking the values of the variables after each performed change. The values of the variables are compared to the values obtained from the Tempura code.

Here is the function for calculating the Y -th value of the **ke_fb[]** array written in Tempura language. The function **itof** returns the float corresponding to an integer number and $del = x_{max} - x_{min}$.

```
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
}.
```

Here is the test we defined to check the results:

```
/* 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)
}
}
}.

```

The function **check** compare the results from the **calc_ke_fb** function and the C program. If they are identical, test passed. The results from the first two iterations when a test is run by the Ana Tempura tool are shown in 2. These test cases pass successfully.

```

AnaTempura: morse
File Edit Check Run
Tempura External Help About
Tempura 5%
run test1().

["fbrke_i","0.000000","1","2","0"].
atime(T)=1
!:: Pass fbrke_i Prog 0.00000 Prop 0.00000

atime(T)=0

["fbrke_i","0.008057","1","2","0"].
atime(T)=1
!:: Pass fbrke_i Prog 0.00806 Prop 0.00806

atime(T)=0

["fbrke_i","0.032227","1","2","0"].
atime(T)=1
!:: Pass fbrke_i Prog 0.03223 Prop 0.03223

atime(T)=0

```

Fig. 2. Execution of test1 in Ana Tempura

5. CONCLUSION AND FUTURE WORK

In this paper we presented the importance of using formal methods for testing and specification of scientific software. A formal specification of the code for solving the bound states of the Morse oscillator well by using ITL and Ana Tempura was described. Also the implementation details and steps were given. Formal specification and description of the scientific software will improve its correctness by reducing the number of errors, especially when a code change is made. The testing also will be more accurate and mathematical model will be provided for the software. We chose ITL and Tempura as a powerful description language and extension to standard predicate logic which includes time-dependent operators. The programs written in Tempura and executed in Ana Tempura tool can communicate with .exe version of the programs. At this moment, there are examples of program codes

written in C and Java. There are no limitations of the number of test cases that can be covered. Tempura language is similar to predicate logic and it is easy to be learned. There are only several rules and categories such as expressions, statements and locations. The hard part for the scientists could be the programming with recursion.

We plan to make a formal specification of the scientific application which is developed within the HP-SEE (High-Performance Computing Infrastructure for South East Europe) project. We want to automate the testing process by creating and running tests using the Ana Tempura tool.

REFERENCES

- E.R. Bittner. 2009. *Quantum dynamics: applications in biological and materials systems*. CRC Press. <http://books.google.mk/books?id=wKrvAAAAAAAJ>
- Antonio Cau, Ben Moszkowski, and Hussein Zedan. 2002. Interval Temporal Logic. (2002). <http://citeseerx.ist.psu.edu/viewdoc/download?rep=rep1&type=pdf&doi=10.1.1.4.6669>
- Edmund M. Clarke and Jeannette M. Wing. 1996. Formal Methods: State of the Art and Future Directions. *ACM Comput. Surv.* 28, 4 (Dec. 1996), 626–643. DOI: <http://dx.doi.org/10.1145/242223.242257>
- De Montfort University. 2004. AnaTempura tool for runtime verification and animation. (2004). <http://www.tech.dmu.ac.uk/STRL/research/software/anatempura.pdf>
- Amin El-kustaban, Ben Moszkowski, and Antonio Cau. 2012. Specification Analysis of Transactional Memory using ITL and AnaTempura. *Lecture Notes in Engineering and Computer Science* 2195, 1 (2012), 176–181.
- J. F. Groote, A. A. H. Osaiweran, and J. Wesselius. *Benefits of applying formal methods to industrial control software*. Technical Report CS-Report 11–04. <http://www.win.tue.nl/~jfg/articles/CSR-11-04.pdf>
- Anthony Hall. 2005. Realising the Benefits of Formal Methods. In *Formal Methods and Software Engineering*, Kung-Kiu Lau and Richard Banach (Eds.). Lecture Notes in Computer Science, Vol. 3785. Springer Berlin Heidelberg, 1–4. DOI: http://dx.doi.org/10.1007/11576280_1
- I. P. Hamilton J. C. Light and J. J. V. Lill. 1985. DVR. *Chem. Phys.* 82, 1400 (1985).
- E. Jaeger. 2010. *Study of the Benefits of Using Deductive Formal Methods for Secure Developments*. <http://books.google.mk/books?id=EU2KMwEACAAJ>
- Bojana Koteska and Anastas Mishev. 2013. Software Engineering Practices and Principles to Increase Quality of Scientific Applications. In *ICT Innovations 2012*, Smile Markovski and Marjan Gusev (Eds.). Advances in Intelligent Systems and Computing, Vol. 207. Springer Berlin Heidelberg, 245–254. DOI: http://dx.doi.org/10.1007/978-3-642-37169-1_24
- Ben Moszkowski. 1985. Executing temporal logic programs. In *Seminar on Concurrency*, Stephen D. Brookes, Andrew William Roscoe, and Glynn Winskel (Eds.). Lecture Notes in Computer Science, Vol. 197. Springer Berlin Heidelberg, 111–130. DOI: http://dx.doi.org/10.1007/3-540-15670-4_6
- Ben Moszkowski and Zohar Manna. 1984. Reasoning in interval temporal logic. In *Logics of Programs*, Edmund Clarke and Dexter Kozen (Eds.). Lecture Notes in Computer Science, Vol. 164. Springer Berlin Heidelberg, 371–382. DOI: http://dx.doi.org/10.1007/3-540-12896-4_374
- Jonathan S. Ostroff and Richard F. Paige. 1998. 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)*. IEEE Computer Society, Washington, DC, USA, 63–. <http://dl.acm.org/citation.cfm?id=554225.828878>
- Carlos Rossi, Manuel Enciso, and Inmaculada P. de Guzmán. 2004. Formalization of UML state machines using temporal logic. *Software and Systems Modeling* 3, 1 (2004), 31–54. DOI: <http://dx.doi.org/10.1007/s10270-003-0029-7>
- Rebecca Sanders and Diane Kelly. 2008. The Challenge of Testing Scientific Software. In *Proceedings of the Conference for the Association for Software Testing*. 30–36.
- Guido Sciacivco, JoseM. Juarez, and Manuel Campos. 2009. Quality Checking of Medical Guidelines Using Interval Temporal Logics: A Case-Study. In *Bioinspired Applications in Artificial and Natural Computation*, Jos Mira, JosManuel Ferrndez, JosR. lvarez, Flix Paz, and F.Javier Toledo (Eds.). Lecture Notes in Computer Science, Vol. 5602. Springer Berlin Heidelberg, 158–167. DOI: http://dx.doi.org/10.1007/978-3-642-02267-8_18
- J. Segal. 2008. Scientists and Software Engineers: A Tale of Two Cultures. In *Proceedings of the Psychology of Programming Interest Group*. <http://www.ppig.org/papers/20th-segal.pdf>
- Stephen F. Siegel and George S. Avrunin. 2004. Verification of MPI-Based Software for Scientific Computation. In *Model Checking Software*, Susanne Graf and Laurent Mounier (Eds.). Lecture Notes in Computer Science, Vol. 2989. Springer Berlin Heidelberg, 286–303. DOI: http://dx.doi.org/10.1007/978-3-540-24732-6_20

- StephenF. Siegel and LouisF. Rossi. 2008. Analyzing BlobFlow: A Case Study Using Model Checking to Verify Parallel Scientific Software. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, Alexey Lastovetsky, Tahar Kechadi, and Jack Dongarra (Eds.). Lecture Notes in Computer Science, Vol. 5205. Springer Berlin Heidelberg, 274–282. DOI : http://dx.doi.org/10.1007/978-3-540-87475-1_37
- I. Sommerville. 2007. *Software Engineering*. Addison-Wesley. <http://books.google.mk/books?id=B7idKfL0H64C>
- M. Umarji, C. Seaman, A.G. Koru, and Hongfang Liu. 2009. Software Engineering Education for Bioinformatics. In *Software Engineering Education and Training, 2009. CSEET '09. 22nd Conference on*. 216–223. DOI : <http://dx.doi.org/10.1109/CSEET.2009.44>
- John A. van der Poll. 2010. Formal methods in software development: a road less travelled. *South African Computer Journal* 45 (2010), 40–52. <http://dblp.uni-trier.de/db/journals/saj/saj45.html#Poll10>
- C. Vecchiola, S. Pandey, and R. Buyya. 2009. High-Performance Cloud Computing: A View of Scientific Applications. In *Proceedings of the 10th International Symposium on Pervasive Systems, Algorithms and Networks (I-SPAN 2009)*. IEEE Computer Society.
- Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui, and John Fitzgerald. 2009. Formal Methods: Practice and Experience. *ACM Comput. Surv.* 41, 4, Article 19 (Oct. 2009), 36 pages. DOI : <http://dx.doi.org/10.1145/1592434.1592436>
- Hussein Zedan, Antonio Cau, Zhiqiang Chen, and Hongji Yang. 1999. ATOM: An object-based formal method for real-time systems. *Annals of Software Engineering* 7, 1-4 (1999), 235–256. DOI : <http://dx.doi.org/10.1023/A:1018942406449>
- Inc. Ziff Davis Publishing Holdings. 1995. PC Magazine. (1995). <http://www.pcmag.com/encyclopedia/term/50872/scientific-application>

Towards a Framework for Usability Testing of Interactive Touchless Applications

SAŠA KUCHAR and KRISTJAN KOŠIČ, University of Maribor

Interaction with user interfaces only with usage of hands and bodies was a few years back still science fiction, but now present reality. Touchless interfaces are slowly becoming mainstream and therefore it is of crucial importance to address them in a concise and standardized way. Usability is an important factor in all software quality models and a key factor during development of interactive applications. The objective of this paper is to address usability challenges we are facing during touchless application design. A conceptual usability case study is proposed with new human-computer interaction factors in mind. Factors like efficiency, ease-of-use, pleasure, fatigue, naturalness, smoothness, responsiveness and accuracy are identified and related to usability scenarios.

Categories and Subject Descriptors: H.5.2 [Information Interfaces and Presentation]: User Interfaces—*Evaluation / methodology*; H.1.2 [Models and Principles]: User/Machine Systems—*Human Information Processing*

General Terms: Human factors, Design, Measurement

Additional Key Words and Phrases: User experience, UX Metrics, Touchless User Interfaces, Human-Computer Interaction, ADORA

1. INTRODUCTION

ISO 9241-210 [2010] defines User Experience (UX) as "a person's perceptions and responses, that result from the use or anticipated use of a product, system or service". It does not include merely the interaction that happens during the use, but all the users' emotions, beliefs, preferences, perceptions, physical and psychological responses, behaviours and accomplishments that occur before, during and after use. Tullis and Albert [2013] extend the definition by adding another characteristic that is: "the users' experience is of interest, and observable or measurable". So according to that definition it is not enough for the user to interact with an interface, but this action has to be measured in some way.

In this paper we elaborate on usability issues with touchless interfaces and review evaluation scenarios as well as UX metrics. We describe the characteristics of touchless UIs in section 3 and introduce a case study on measuring usability of gesture interfaces in section 4. Finally, section 5 presents conclusions and discusses future directions of this work.

2. USER EXPERIENCE AND USABILITY

UX must not be confused with Usability as usability is a narrower term. Usability is considered the ability of the user to use the thing to carry out a task successfully: "The extent to which a product can be used by specified users to achieve specified goals with **effectiveness**, **efficiency** and **satisfaction** in a specified context of use" [9241-11 1998]. UX looks at the individual's entire interaction with the

Author's address: S. Kuhar, Faculty of electrical engineering and computer science, University of Maribor, Smetanova ul. 17, 2000 Maribor, Slovenia; email: sasa.kuhar@um.si; K. Košič, Faculty of electrical engineering and computer science, University of Maribor, Smetanova ul. 17, 2000 Maribor, Slovenia; email: kristjan.kosic@um.si;

Copyright © by the paper's authors. Copying permitted only for private and academic purposes.

In: Z. Budimac, T. Galinac Grbac (eds.): Proceedings of the 3rd Workshop on Software Quality, Analysis, Monitoring, Improvement, and Applications (SQAMIA), Lovran, Croatia, 19.-22.9.2014, published at <http://ceur-ws.org>.

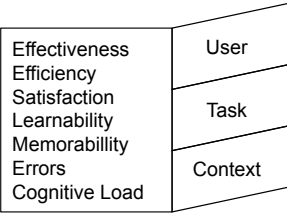


Fig. 1. PACMAD Usability model [Harrison et al. 2013]

thing, as well as thoughts, feelings, and perceptions, that result from that interaction [Tullis and Albert 2013].

When discussing usability, it is important to distinguish between summative and formative usability evaluation. Formative usability has strong ties to the practice of iterative design — building something, checking to see where it could be improved, improving it, and trying again. Summative evaluations emphasize the importance of effectiveness and efficiency in the context of use and the subjective metric of satisfaction [Lewis 2014]. The main goal of summative evaluation study is to evaluate whether people can use a product for its intended purpose effectively, efficiently, and with a feeling of satisfaction. The formative evaluation study reveals the presence of usability when there is absence of usability problems.

Nielsen [1993] defined five attributes of usability: (i) efficiency (relative to the accuracy and completeness with which users achieve goals), (ii) satisfaction (freedom from discomfort and positive attitudes towards the use of the product), (iii) learnability (the system should be easy to learn so that the user can rapidly start getting work done with the system), (iv) memorability (the system should be easy to remember so that the casual user is able to return to the system after some period of not having used it without having to learn everything all over again) and (v) errors (the system should have a low error rate, so that users make few errors during the use of the system and that if they do make errors they can easily recover from them).

Harrison et al. [2013] combined the attributes from ISO 9241-11 [1998] and Nielsen [1993] and added another interesting factor, that is **cognitive load**. Cognitive load refers to the amount of cognitive processing required by the user to use the application. Harrison’s PACMAD model (figure 1) derives from mobile devices, where he argues that users of mobile applications may be performing additional tasks, such as walking, while using the mobile device. In our opinion the model could be well applied to touchless user UIs as those require from a user to combine usual interaction tasks with speaking and moving. In touchless UIs user needs to use predefined gestures and voice control simultaneously while executing wanted tasks and this can raise the mental load. Therefore it is important that the aspect of cognitive load is included in usability studies of touchless interfaces. More about touchless UIs is described in section 3.

All previously mentioned models recognise three factors that can affect usability of an application. These are: (i) user (users physical limitations, their knowledge and previous experience), (ii) task (goal the user is trying to accomplish) and (iii) context of use (environment in which the user will use the application including physical location and interaction with other people and objects). All of these are eligible for usability of touchless UIs.

2.1 How to evaluate UX

Use experience has to be measured in order to evaluate whether it is good or bad, to discover problems and opportunities for improvement, or to compare different UIs. Many metrics are available for the

Table I. Eleven common usability scenarios and the metrics that may be most appropriate for each.

Usability Study Scenario	Task Success	Task Time	Errors	Efficiency	Learn-ability	Issues-based Metrics	Self-reported Metrics	Behavioral & Physiological Metrics	Combined & Comparative Metrics	Live Website Metrics	Card-Sorting Data
1. Completing a transaction	x			x		x	x			x	
2. Comparing Products	x			x			x		x		
3. Evaluating frequent use of the same product	x	x		x	x		x				
4. Evaluating navigation and/or information architecture	x		x	x							x
5. Increasing awareness							x	x		x	
6. Problem discovery						x	x				
7. Maximizing usability for a critical product	x		x	x							
8. Creating an overall positive experience							x	x			
9. Evaluating the impact of subtle changes										x	
10. Comparing alternative designs	x	x				x	x		x		
11. Cognitive load	x	x					x	x			

The table is adapted from [Tullis and Albert 2013] and is supplemented with cognitive load scenario.

evaluation of UX. Metrics add structure to the design and evaluation process, give insight into the findings and provide information to the decision makers, they offer a way to estimate the number of users likely to experience a problem, are a key ingredient in calculation of ROI as well as can reveal patterns that are difficult or even impossible to see. For that UX metrics need to be observable (directly or indirectly), quantifiable (turned into a number or counted in some way), and they have to measure some aspect of the user experience [Tullis and Albert 2013].

Many different metrics are available for different evaluations. When evaluating UX one has to consider the goals of the study, the technology that is available to collect the data, and budget as well as time that are available for conducting evaluation. In table I we present ten common usability scenarios and the metrics that may be most appropriate for each, adapted from [Tullis and Albert 2013]. We added 11th scenario: cognitive load, as we believe it is important for evaluation of touchless UIs.

For details about the scenarios and metrics see [Tullis and Albert 2013]. At this point we describe only the newly added scenario.

2.2 Cognitive Load measurement for evaluation of UX

Cognitive load theory (CLT) describes the relationship between the capacity of working memory and the cognitive demands of a particular task [Anderson 2012]. It is based on an idea, that cognitive capacity in working memory is limited and that if a mental task requires too much capacity knowledge acquisition and reasoning will be hindered [Jong 2009]. One solution to this is to design a UX that optimizes the use of working memory capacity and avoids cognitive overload.

CLT distinguishes between three types of cognitive load: (i) intrinsic (the inherent difficulty of the problem at hand), (ii) extraneous (generated by the representation of the content presented to the user for interpretation and action) and (iii) germane (imposed by learning a new task) [Anderson 2012]. Extraneous, intrinsic and germane cognitive load are modelled to be additive: a reduction of extrane-

ous cognitive load frees working memory capacity that can be used for germane learning processes [Hollender et al. 2010].

The amount of extraneous load due to software use is influenced by the complexity of the software, a suboptimal software design according to traditional usability goals, and the expertise of the learner with regard to the use of the software. Load can be lowered by designing highly usable software applications and by training learners to use the software [Hollender et al. 2010].

Cognitive load can be measured in different ways. Most common measurements include task completion time and accuracy, NASA-TLX test (a survey with subjective responses), EEG-based measurement (determining cognitive load magnitude by analysing the temporal, spectral, and spatial patterns of brain activity), pupil dilation, eye tracking and blinking measurement, galvanic skin response, and heat flux [Anderson 2012; Chen et al. 2011; Haapalainen et al. 2010].

3. TOUCHLESS USER INTERFACES

Touchless user interfaces require devices, that can either execute or sense interactive behaviour, where the interaction happens without mechanical contact between the human and any part of artificial system. Touchless interaction can be multimodal, in which case the interactive behaviour produces simultaneous events in the visual modality (colour, form, or position change), in the auditory modality (speech, sounds), or in the olfactory modality (odors) [de la Barré et al. 2009]. Voice control can be realized by recording the voice with a microphone and processing it through dedicated algorithms. Body gestures can be detected in different ways from using wearable sensors to environmental sensors [Jalaliniya et al. 2013].

Touchless UIs are suppose to remove the burden of physical contact with an interactive system and make interaction pleasurable [de la Barré et al. 2009]. To achieve that one must carefully combine characteristics of physical and digital world and must try to produce a solution with "natural" Human-Computer Interaction (HCI). One must also consider the fact that hand movement is not equal to gesture, as gesture is a body movement which is being performed with the perceivable intention to express something [de la Barré et al. 2009]. For the user to learn all the gestures and voice controls, that are implemented in a certain UI, a built-in tutorial is a welcomed feature in touchless UIs.

4. CASE STUDY DESIGN: MEASURING USABILITY OF GESTURE INTERFACES DURING SURGERY WITH ADORA

According to Madan and Dubey [2012] usability is the most widely used concept in software engineering field and defines the software system's demand and use. Demand for software quality and usability is increasing and there are several usability models, that can be used and tailored to our needs (for more on usability models see Madan and Dubey [2012]).

4.1 Usability and gesture interfaces

The specifics of HCI need additional factors that will help us to successfully implement a usability study with gesture interaction in mind. Touchless HCI software interacts with users by using gestures and voice commands that are tied to gesture recognition engine. Factors like efficiency, ease-of-use, pleasure, fatigue, naturalness, smoothness, responsiveness, and accuracy should be investigated in detail. Investigation should be based through simple and complex tasks. According to Farhadi-Niaki et al. [2013] gesture based systems cause more fatigue and appear less natural then finger gestures, however factors such as time, overall satisfaction, and easiness were not affected.

Gesture recognition is a rather new field of HCI, so it is normal that problems are present. The main issues are usually: (i) lack of standardization gestures, (ii) lack of cues, (iii) inability to discover operations and (iv) requiring memorization of the player (head memory). These problems were classified by

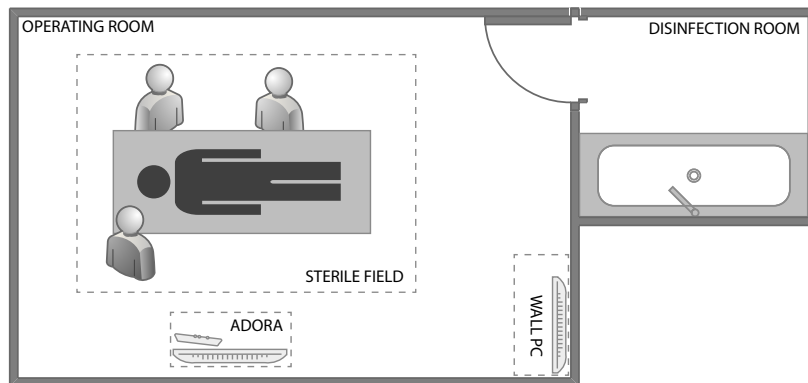


Fig. 2. Touch-less surgery with ADORA. Placement in an operating room. Problem of accessing data and renewed disinfection.

Norman et al. [2010] where he also argued that gesture based interfaces are a **step backwards in usability** due to lack of:

- (1) **Visibility** - available gestures at each moment are not clear, neither speeds or precision of the movement.
- (2) **Feedback** - there is not enough feedback for the user during the gesture. The user does not know if the action was because of the correct gesture or something failed in the detection.
- (3) **Consistency and Standards** - there are no standards that would define consistency of HCI menus or standardized gestures.
- (4) **Discoverability** - the user must know all the gestures in advance (head memory), or there is no feedback that would help him to connect gestures with application outcomes.
- (5) **Reliability** -

”...Accidental gesture activation is common in gestural interfaces, as users happen to touch something they didn’t mean to touch. Conversely, frequently users do intend to touch a control or issue a gestural command but nothing happens because their touch or gesture was a little bit off. Since gestures are invisible, users often do not know that they made these mistakes” [Norman et al. 2010].

This is especially the case with Microsoft Kinect camera.

Therefore all the above mentioned issues need to be properly addressed in UX evaluation after they have been carefully studied, planned, and implemented in a system.



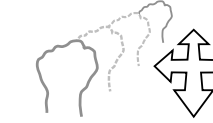
4.2 ADORA solution

ADORA¹ is an interactive physician’s assistant, that enables presentation of information about a patient before and during surgical procedures. It offers a comprehensive and integrated natural user interface experience for physicians. With its use of contact-free interaction it shortens the duration of surgeries and indirectly affects the environmental and economic aspects of healthcare.

Touchless methods of HCI (gestures and voice support) have been integrated into ADORA solution during development. Before using touchless assistant physicians had to leave the patient and the sterile field in order to access critical patient data, that was accessible through the wall PC. In order to

¹<http://www.adora-med.com>.

Table II. Most common used gestures in ADORA

Push	Grab and move	Grab and move in depth
		

continue with the surgery they had to disinfect, change gloves, gown and mask), memorize the picture and then return to the patient (see figure 2). This takes time and causes additional stress for the doctor. ADORA delivers patient data in a touchless manner, so that the surgeon can stay by the patient while viewing the data. Touchless interaction was developed with the help of a Microsoft Kinect sensor.

The solution was designed together with the users, physicians. One of the main requests was a minimal set of gestures, so that they can use the solution in the most possible natural way. As the surgeon has always active hands during surgery, all the gestures and their sets are designed in a one-handed way. Table II shows most common used gestures in ADORA.

4.3 Experiment design and procedure

The most important thing during planning a usability study is understanding users and the goals they are trying to accomplish. The user and his expectations from the solution define core building blocks of the study. In our case study a user is a physician in an operating room who is performing long complex surgeries. His personal goal is to successfully complete a surgery with as little distraction possible.

ADORA solution is already implemented and finalized, therefore a summative usability study is in order. Tullis and Albert [2013] define summative usability as answers to the following questions: (i) Did we meet the usability goals of the project?, (ii) What is the overall usability of the product?, (iii) How does our product compare against the competition? and (iv) Have we made improvements from one product release to the next?. Main goal of summative usability study is to evaluate how well a product or piece of functionality meets its objectives.

Experiment design reflects the scenarios selected from table I and the factors that are specific for HCI and gesture interfaces. The experiment will consists of three parts. Pre-test survey: to gather participant's data that will be used during the main part of experiment. Practice and test sessions: the core of the experiment will consist of tasks completion rate, possible errors and factors related to them. The final part will be executed with the post-task questionnaire.

In the main part of the evaluation, each participant will be instructed to perform a defined task using gestures and voice commands to control ADORA solution with the help of Kinect sensor. Tasks will include normal operations that surgeons are otherwise used carrying out on a wall PC.

Task were based on the Ux scenarios in table I. Scenario one, four and seven were addressed with selected tasks and post-task questionnaires. At this point cognitive load scenario is not included.

Tasks will be split into simple and advanced sets. For example a simple set of tasks would be selecting a surgery (pushing a button with gesture) and loading a set of medical images for the patient (a combined set of gestures). Advanced set of tasks will include changing a view, loading multiple image series into different views, combining the views and manipulation of medical images such as point based zoom and image adjusting (combined operation of brightness and contrast merged into one gesture). Table III list a set of tasks surgeon will be performing during experiment.

There will be two groups of participants (surgeons). First group will be educated about functionalities and gestures that are available in ADORA, while the second group will identify gestures with the help of a tutorial that is available in the solution. Group data will be used to determine if pre-education of

Table III. Tasks a surgeon will be performing during second phase of study

Task	Gesture	Voice	Level
Login	Push button	Y	basic
Select operation	Drag to selected surgery and push	N	advanced
Change view	A combined set	Y	basic
Load a set of images	Interactive push with feedback	N	basic
Combine set of images	Advanced set	N	basic
Point zoom to a defined spot	Grab and pull/push	N	advanced
Adjust image to specified level	Grab and move	N	advanced
Lock position	Move and push	Y	basic

Advanced gestures are combined with domain knowledge that surgeons need during task execution.

the users affects the overall experience, or if the built-in tutorial is enough to gain needed operational knowledge.

During task execution data will be gathered according to Fitts' Law Test [Zhai 2004] in terms of completion time, errors and throughput. Completion time will be measured in three iterations and then average will be calculated of the multi-directional Fitt's law tasks. A post analysis of measured times will be done with Scheffe criterion for significance [Fleiss 1999]. Similar analysis will be made for throughput and error.

In the final phase, after task completion, users will be asked to complete a survey containing a Device Assessment Questionnaire suggested by ISO 9241-9 [2000]. All questions will be ranked with a 7-point Likert scale, from strongly agree to strongly disagree and will be fine-tuned with inclusion of HCI factors mentioned above. Only a combination of questionnaires, performance metrics, and inclusion of HCI factors can give a comprehensive and valid picture for several reasons: (i) users might be influenced not to honestly report their experience, (ii) a combination will give us more insight what system factors influenced noteworthy user ratings, (iii) for most of the quality of experience aspects described, there exist no valid and reliable metrics for the case of gesture interfaced systems, so a mixture can help to interpret the results better [Wechsung et al. 2012].

5. CONCLUSION

In this paper terms and definitions of Usability, User experience and Human Computer Interaction were challenged all together. There are already many models that define usability and user experience, but none of them has yet been tailored to the needs and challenges of gestural interfaces. While gestural interfaces are yet to become mainstream, a lot of them can be found in the gaming world - where touchless devices have already been well accepted.

Having a natural mapping between the body actions and the reactions on the screen gives very positive reactions and in order to keep the positive interaction flow with the users, further research is needed in the gestural interfaces domain. A standardized framework is needed, that will define consistency and standards for evaluation of gestural interfaces and that will include the aspect of cognitive load.

REFERENCES

- ISO 9241-11. 1998. ISO 9241-11:1998 - Ergonomic requirements for office work with visual display terminals (VDTs) – Part 11: Guidance on usability. (1998). http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=16883
- ISO 9241-210. 2010. ISO 9241-210:2010 - Ergonomics of human-system interaction – Part 210: Human-centred design for interactive systems. (2010). http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=52075

- ISO 9241-9. 2000. ISO 9241-09:2000 - Ergonomic requirements for office work with visual display terminals (VDTs) – Part 9: Requirements for non-keyboard input devices. (2000). http://www.iso.org/iso/catalogue_detail.htm?csnumber=30030
- Erik W. Anderson. 2012. Evaluating visualization using cognitive measures. In *Proceedings of the 2012 BELIV Workshop on Beyond Time and Errors - Novel Evaluation Methods for Visualization - BELIV '12*. ACM Press, New York, New York, USA, 1–4. DOI: <http://dx.doi.org/10.1145/2442576.2442581>
- Siyuan Chen, Julien Epps, and Fang Chen. 2011. A comparison of four methods for cognitive load measurement. In *Proceedings of the 23rd Australian Computer-Human Interaction Conference on - OzCHI '11*. ACM Press, New York, New York, USA, 76–79. DOI: <http://dx.doi.org/10.1145/2071536.2071547>
- R de la Barré, Paul Chojecski, and Ulrich Leiner. 2009. Touchless interaction-novel chances and challenges. In *Proceedings of the 13th International Conference on Human-Computer Interaction*, Julie A. Jacko (Ed.). Springer-Verlag Berlin, Heidelberg, 161–169. DOI: http://dx.doi.org/10.1007/978-3-642-02577-8_18
- Farzin Farhadi-Niaki, S.Ali Etemad, and Ali Arya. 2013. Design and Usability Analysis of Gesture-Based Control for Common Desktop Tasks. In *Human-Computer Interaction. Interaction Modalities and Techniques*, Masaaki Kurosu (Ed.). Lecture Notes in Computer Science, Vol. 8007. Springer Berlin Heidelberg, 215–224. DOI: http://dx.doi.org/10.1007/978-3-642-39330-3_23
- Joseph L. Fleiss. 1999. *The Parallel Groups Design*. John Wiley & Sons, Inc., 46–90. DOI: <http://dx.doi.org/10.1002/9781118032923.ch3>
- Eija Haapalainen, SeungJun Kim, Jodi F. Forlizzi, and Anind K. Dey. 2010. Psycho-physiological measures for assessing cognitive load. In *Proceedings of the 12th ACM international conference on Ubiquitous computing - Ubicomp '10*. ACM Press, New York, New York, USA, 301. DOI: <http://dx.doi.org/10.1145/1864349.1864395>
- Rachel Harrison, Derek Flood, and David Duce. 2013. Usability of mobile applications: literature review and rationale for a new usability model. *Journal of Interaction Science* 1, 1 (Dec. 2013), 1–16. DOI: <http://dx.doi.org/10.1186/2194-0827-1-1>
- Nina Hollender, Cristian Hofmann, Michael Deneke, and Bernhard Schmitz. 2010. Review: Integrating Cognitive Load Theory and Concepts of Human-computer Interaction. *Comput. Hum. Behav.* 26, 6 (Nov. 2010), 1278–1288. DOI: <http://dx.doi.org/10.1016/j.chb.2010.05.031>
- Shahram Jalaliniya, Jeremiah Smith, Miguel Sousa, Lars Bütthe, and Thomas Pederson. 2013. Touch-less interaction with medical images using hand & foot gestures. In *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication - UbiComp '13 Adjunct*. ACM Press, New York, New York, USA, 1265–1274. DOI: <http://dx.doi.org/10.1145/2494091.2497332>
- Ton Jong. 2009. Cognitive load theory, educational research, and instructional design: some food for thought. *Instructional Science* 38, 2 (Aug. 2009), 105–134. DOI: <http://dx.doi.org/10.1007/s11251-009-9110-0>
- James R. Lewis. 2014. Usability: Lessons Learned ... and Yet to Be Learned. *International Journal of Human-Computer Interaction* 30, 9 (June 2014), 663–684. DOI: <http://dx.doi.org/10.1080/10447318.2014.930311>
- Ankita Madan and Sanjay Kumar Dubey. 2012. Usability evaluation methods: a literature review. *International Journal of Engineering Science and ...* 4, 02 (2012), 590–599. <http://www.ijest.info/docs/IJEST12-04-02-143.pdf>
- Jakob Nielsen. 1993. *Usability Engineering*. Morgan Kaufmann. 362 pages. <http://www.nngroup.com/books/usability-engineering/>
- Donald A Norman, Jakob Nielsen, and Nielsen Norman Group. 2010. *Gestural Interfaces : A Step Backward In Usability*. (2010).
- Thomas Tullis and William Albert. 2013. *Measuring the User Experience, Second Edition: Collecting, Analyzing, and Presenting Usability Metrics*. (July 2013). <http://dl.acm.org/citation.cfm?id=2526258>
- Ina Wechsung, Klaus-Peter Engelbrecht, Christine Kühnel, Sebastian Möller, and Benjamin Weiss. 2012. Measuring the Quality of Service and Quality of Experience of multimodal human-machine interaction. *Journal on Multimodal User Interfaces* 6, 1-2 (2012), 73–85. DOI: <http://dx.doi.org/10.1007/s12193-011-0088-y>
- Shumin Zhai. 2004. Characterizing computer input with Fitts' law parameters—the information and non-information aspects of pointing. *International Journal of Human-Computer Studies* 61, 6 (Dec. 2004), 791–809. DOI: <http://dx.doi.org/10.1016/j.ijhcs.2004.09.006>

Techniques for Bug–Code Linking

GORAN MAUŠA, PAOLO PERKOVIĆ, TIHANA GALINAC GRBAC and IVAN ŠTAJDUHAR,
Faculty of Engineering, University of Rijeka

Diverse research groups have identified that analyzing the bugs-to-code fix links may generate many interesting theories. However, this kind of datasets are usually not easily available from the software development projects. Usually, the bug tracking and the related code fix activities are divided into separate processes and use separated repositories thus complicating the linking of the data. Numerous techniques have been developed but we still lack the empirical data collected with standard data collection procedures.

In this paper we examine the effectiveness of the bug linking techniques. In particular we evaluate the proposed technique based on the regular expressions and compare its effectiveness with other known bug linking techniques. Our case study shows that the proposed technique is equally effective as some other already proposed techniques in eliminating the linking bias we identified some differences that may have influence on the linking precision. However, the technique is not addressing data quality issues that may be present in software repositories.

Categories and Subject Descriptors: D.2.5 [**Software Engineering**]: Testing and Debugging—*Tracing*; D.2.9 [**Software Engineering**]: Management—*Software quality assurance (SQA)*; H.3.3 [**Information Storage and Retrieval**] Information Search and Retrieval

Additional Key Words and Phrases: Bug – code linking, software, defect, open source repositories

1. INTRODUCTION

Analyzing bugs-to-code fix links may generate many interesting theories. For example, new empirical findings aiming to contribute the theory of fault distributions may have significant influence on future software development methods and practices [Grbac et al. 2013; Petrić and Grbac 2014; Grbac and Huljenic 2014]. Since bugs and code commits are usually stored in different repositories with different information needs their links have to be mined. Thus, such retrieved datasets suffer from data quality as well as data collection bias. Data quality is very much affected by software developers responsible for filling the data and missing or wrongly inserted data in software repositories are a result of human error. On the other hand, retroactive data collection and linking between different repositories represent another problem that is well known as linking bias. A Link may be incorrectly established or missing. The reason may be in ineffectiveness of linking technique employed and/or weak understanding of underlying complex software development interactions.

As part of our research in the fault distribution of complex systems we want to perform extensive explorative analysis on bug–code datasets [Galinac Grbac et al. 2013]. As software defect datasets available for such analysis have been criticized by number of authors we were motivated to develop a

The work presented in this paper is supported by the University of Rijeka research grant Grant 13.09.2.2.16.

Author's address: G. Mauša, Faculty of Engineering, Vukovarska 58, HR–51000 Rijeka, Croatia; email: gmausa@riteh.hr; P. Perković, student at the Faculty of Engineering, Vukovarska 58, HR–51000 Rijeka, Croatia; email: pperkovic@riteh.hr; T. Galinac Grbac, Faculty of Engineering, Vukovarska 58, HR–51000 Rijeka, Croatia; email: tgalinac@riteh.hr; I. Štajduhar, Faculty of Engineering, Vukovarska 58, HR–51000 Rijeka, Croatia; email: istajduh@riteh.hr

Copyright © by the paper's authors. Copying permitted only for private and academic purposes.

In: Z. Budimac, T. Galinac Grbac (eds.): Proceedings of the 3rd Workshop on Software Quality, Analysis, Monitoring, Improvement, and Applications (SQAMIA), Lovran, Croatia, 19.-22.9.2014, published at <http://ceur-ws.org>.

tool that would be based on the numerous use open source available repositories build as much as possible bug-to-code fix datasets. This paper deals with definition of such a tool that we will refer to as BuCo (Bug-Code) tool.

In the last few years, research on bug-fix datasets has progressed well, since the number of open source projects and using of open source project repositories grow and reported many important results. The popularization of open source software enabled individuals to introduce different ways of software development and code analysis. However, datasets bias still remains as a huge issue for wider generalization of the results and development of sound theories. The bug-code fix linking process is extremely hard task and there is still no standard protocol that would be applicable for wider community and numerous software development repositories. Different linking techniques have been proposed in order to minimize the linking bias and some even try to minimize the impact of data quality issues. However, we believe that different linking strategies may not be equally effective for all kind of repositories and software projects. Obviously, proposing the general dataset collection procedures and linking techniques may not be relevant. Instead of proposing the best performing one for open source repositories we review several linking techniques and evaluated their effectiveness on different datasets. We discuss its applicability for data collection from complex open source software projects.

2. RELATED WORK

Many researchers mine open source software development repositories trying to collect bug-fix datasets. One of the most commonly applied technique is using regular expressions [Bachmann et al. 2010; Fischer et al. 2003; Schrter et al. 2006; Śliwerski et al. 2005; Čubranić and Murphy 2003]. It is a sequence of characters that forms a search pattern, most commonly used for string matching used for information retrieval [Baeza-Yates and Ribeiro-Neto 1999]. Drawing on the idea introduced by a neuroscientist and a logician in the early 1940s, it was formally introduced by a mathematician 15 years later, relating the concept to finite automata [Kleene 1956]. First implementation of a regular expression compiler was developed in 1968 by the Unix pioneer, Ken Thompson [Thompson 1968], and has since been used in various problem domains. A regular expression consists of both regular and meta-characters. Both types are combined to form an expression for identifying the pattern sequence (sequence of interest) in a sequence of characters. In other words, the pattern sequence is an expression representing pre-scribed targets in the most concise and flexible way to direct the automation of character sequence processing.

A regular expression processor translates into a nondeterministic finite automaton, which is then run on the sequence of characters to recognize subsequences that match the regular expression [Baeza-Yates and Ribeiro-Neto 1999]. The most commonly used linking technique is based on regular expression and is searching in source code commit messages for a specific keyword such as *fixed* or *bug* and *bugID* (such as 42233) usually using form of regular expression [Bachmann et al. 2010].

Many researchers aiming to develop the most effective linking technique and to understand dataset bias and data quality issues have invested huge efforts to develop a reliable dataset for the benchmarking purposes. As part of the study published with [Bachmann et al. 2010] the authors work with an experienced *Apache* developer who helped them to manually develop an *ground truth* dataset. The main focus of the study was to understand the quality of the data provided in the open source software development repositories. They found out that many bug fixes are not identified within the commit messages or even not reported within the bug tracking repository and that this finding might be a serious threat to studies based on the such datasets. However, the sample size they had was not large enough to make any statistical conclusions but provide some reasonable evidence to threats to external validity of studies performed on linked datasets from open source repositories. One solution to the data quality problem they see in improving the linking process with information available from the whole

social eco–system available such as revision control system, bug tracking database, mailing systems, email discussions, etc.

That finding motivated a number of studies aiming to improve the linking techniques. Improvements may be classified into two directions; direction that involves additional information available from the open repositories and direction that introduces fancy algorithms for the link prediction purposes. One very good example is ReLink, a tool proposed in [Wu et al. 2011] that offered a 2–cycle linking process. In the first round the traditional linking technique based on the regular expression as mentioned above has been extended with new features as follows:

- bug fixing time that is close to the change–commit time,
- the change logs and the bug report share textual similarity, and
- that the developers responsible for a bug are typically the committers of the bug–fixing change.

In the second round, based on the linking dataset in the first round as training sample and some learning rules, predicts additional linking data items. They compared their results with the ground truth file prepared by [Sureka et al. 2011]. As reported in [Wu et al. 2011] they obtained the consistent results with [Sureka et al. 2011] for the Apache project. Also, they conclude that many open source projects hosted by Google are similar and follow the features they implemented in the ReLink. On the other hand, they identified that linking improvements were not equally effective for all datasets. For example linking improvement for Apache was marginal compared to other two Android projects in the study. The reason for this result lies in relatively good data quality of Apache compared to Android projects. We assume that for bigger and complex open source projects the data quality probably tends to be maximized for several reasons. Firstly, the project takes longer and community has time to learn ways of working and less human error happens and another reason is that the samples are getting bigger and thus marginalize errors.

Recently published study [Bissyande et al. 2013] has validated their results and assessed the effectiveness of ReLink quantitatively and qualitatively. Again, they confirmed that in the different project setting the effectiveness of ReLink improvements may be marginal but also they show that these improvements may lead to new errors thus significantly influencing the linking bias. They also provided a benchmark dataset collected from Apache Software Foundation with input and output files provided for 10 projects.

3. EXPERIMENT

Our goal is to identify and explore the most effective linking technique across number of different environments. In this study we define an experiment for that purpose. The ReLink tool, as a good example of combining the simple search with advanced prediction techniques is chosen for this experiment. However, it was only compared with other similarly complex approaches as well as the ground truth dataset. Moreover, because of the input data it provides to open community we decided to investigate how well does it perform comparing to the simpler search approaches. That is why we plan to conduct our experiment in several stages:

- (1) **Analysis 1:** Compare the ReLink tool with the simple search on Apache HTTPD dataset given by ReLink as presented in figure 1
- (2) Evaluate the results, manually investigate the incorrect links and construct a regular expression search criterion
- (3) **Analysis 2:** Compare the ReLink tool with the regular expression search using the same Apache HTTPD dataset as presented in figure 1

- (4) **Analysis 3:** Compare the ReLink tool and the regular expression search with the benchmark dataset for the Apache Opennlp project as presented in figure 2

The **analysis 1** consists of comparing the ReLink tool with the basic idea of linking bugs and commits searching for the Bug ID number in the commit messages. We believe this approach to be the most reliable one. The only question is how to search for the Bug ID effectively. That is why our first approach to the linking was as simple as that. The only improvement we added was to search only for the latest commit that contains the bug ID in order to discard all the duplicated entries, bug fixing attempts and workarounds. The input bugs are taken from the ReLink webpage¹ while the source code is taken from the Github repository of the Apache HTTPD project². These inputs are given to the BuCo Analyzer tool and the linking is done using simple search mechanism.

The results of **analysis 1** will indicate the weaknesses of simple search. The manual investigation of the incorrect links will look for regularities and consistent errors in order to yield a criterion for more sophisticated search. We will construct the regular expression based on this criterion and repeat the analysis. For the **analysis 2** we use the same Apache HTTPD project with the same inputs as in the **analysis 1** (figure 1).

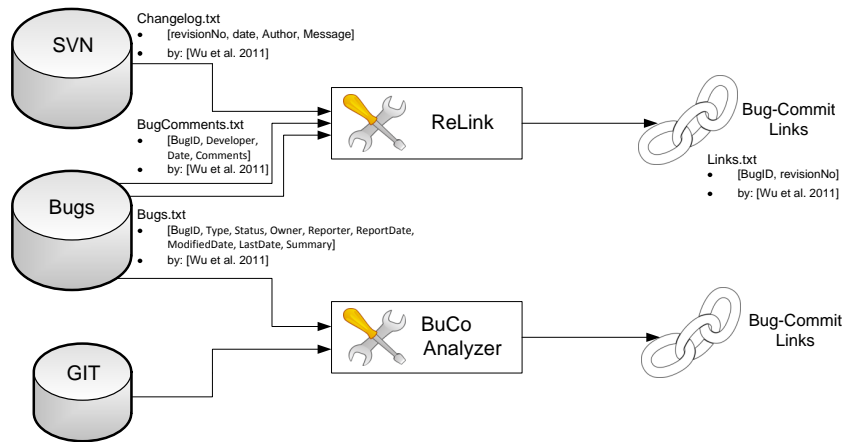


Fig. 1. The Source and Structure of Inputs and Outputs for **Analysis 1** and **2**

The **analysis 3** compares the ReLink tool and the regular expression in a controlled environment of a benchmark dataset. The Opennlp project is analyzed and given by [Bissyande et al. 2013]. The goal of the Opennlp data is to provide a benchmark dataset for comparison of studies such as this one. It offers the subset of bugs, the source code history and the correct links between the bugs and commits made by developers. Furthermore, it offers the comments on all of the given bugs extracted from bug tracking repository and the bug-commit links made by the ReLink tool. The bugs for the Opennlp project are taken from the benchmark dataset³. The source code is taken from the Github repository of the Apache Opennlp project⁴. These inputs are given to the BuCo Analyzer tool and the linking is done using regular expression search mechanism. The details are presented in figure 2.

¹<https://code.google.com/p/bugcenter/wiki/ReLink>

²<https://github.com/apache/httpd.git>

³<http://momentum.labri.fr/bugLinking/>

⁴<https://github.com/apache/opennlp>

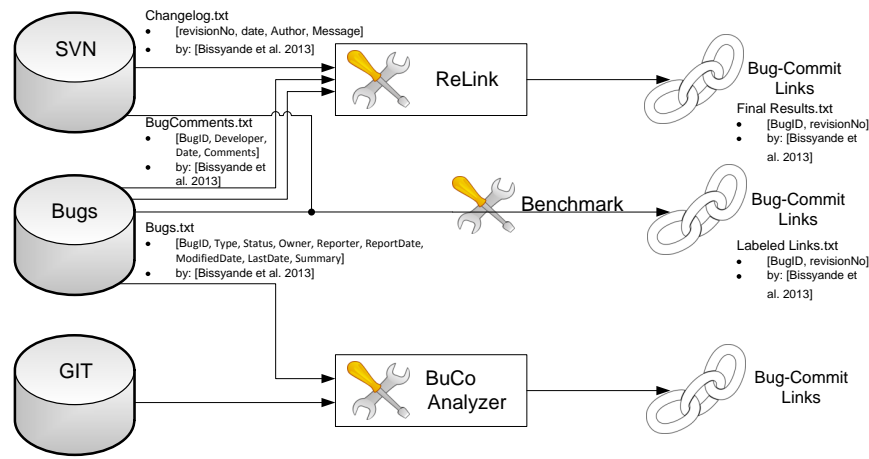


Fig. 2. The Source and Structure of Inputs and Outputs for **Analysis 3**

3.1 Approach to Data Collection

One of the first challenge encountered in our research was to find the appropriate tools for such a demanding task of data collection. We managed to find no tool that could provide us with the source code analysis of software product metrics and the number of bugs contained within every software module. That is why we developed a tool of our own and named it the Bug Code (BuCo) Analyzer. Its goal is to collect software fault's data and find its dependencies in the most efficient way possible. The tool is based upon the following technologies: Apache HTTP server, MySQL relational databases, Git and Subversion source control tools, Python and data mining. They were chosen after their reliability and experienced gained from using them in past projects.

The local database within the tool is constructed so that it can contain the bug information details, the links between the bug and the source code changes (commits) and the list of source code modules and all of their metrics for projects of three major open source communities: Eclipse, Mozilla and Apache. The bug details can be downloaded from the Bugzilla repository into our database and the complete source code history can be downloaded through the tool interface and stored locally as a GIT repository. The BuCo Analyzer also offers interface to other tools. One such tool is the ReLink. Its task is to establish connections between the bugs and the commits. However, that is just one of the techniques the tool offers for this linking process. The complete architecture of the BuCo Analyzer tool is presented in figure 3

As mentioned earlier, our intention is to analyze large and complex software products with long lasting evolution, i.e. a great number of releases. All the analyses then become demanding and slow. To optimize and gain every little boost in terms of speed some smart thinking and known techniques are used in the creation. MySQL databases which would store the data needs to be passed through normalizations steps and has to be modeled after the tool's needs. To further speed up the process repositories are downloaded and analyzed locally instead of analyzing them remotely over the Internet. Other optimizations are done by building efficient search patterns and compiling the patterns when they are necessary.

The tool was being improved iteratively. The first and the most notable aspects is the introduction of pattern matching using regular expressions. This version is able to collect all commits for every bug ID found and their respective messages and implements the collection of bug IDs and downloads

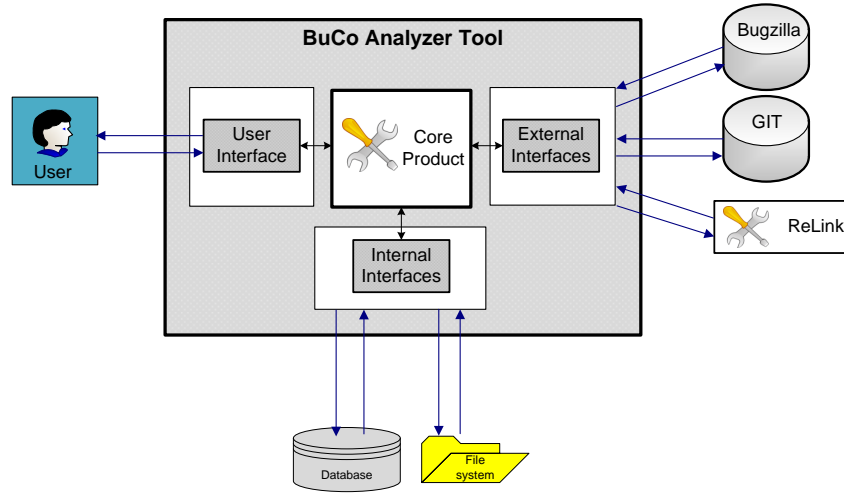


Fig. 3. The BuCo Analyzer Tool Architecture

of repositories for different foundations. The latest tool version offers a high credibility level for the bug-commit links and the software product metrics associated to the software modules.

3.2 Results

The results of **analysis 1** are presented in table I. The ReLink tool performed the linking with the same bugs but with different source code repository, the SVN. The difference between the two repositories is obvious in the number of commits they contain and it reflects on the output results as well.

Table I. Linking Bugs and Commits Using Simple Search, Regular Expression and The ReLink Tool

Analysis	Input			Linking Method	Output			
	Source	Commits	Bugs		Links	Commits	Files	Bugs
1	SVN + Bugs by Relink	43867	673	ReLink	1014	957	1061	673
	GIT + Bugs by Relink	26287	673	Simple search	598	556	993	598
2	SVN + Bugs by Relink	43867	673	ReLink	1014	957	1061	673
	GIT + Bugs by Relink	26287	673	Regular Expression	703	664	495	621
3	SVN + Bugs by benchmark	847	100	Benchmark	127	125	141	81
	SVN + Bugs by benchmark	847	100	ReLink	115	113	132	76
	GIT + Bugs by benchmark	847	100	Regular Expression	128	126	141	81

The results of the simple search were then manually analyzed, compared to the ReLink results and validated. We focused on finding the links that the ReLink and the simple search had in common in order to evaluate the general appropriateness of the approach and on the links that are different in order to discover the simple search weaknesses. The results given in table II indicate that 74.1% of the links are equal to the ones made by the ReLink and all of them were correct ones. The 20.1% of the links that were not equal to the ones made by the ReLink and were incorrect presented a valuable subset to discover patterns and consistent errors in linking process.

The evaluation and manual validation of the results revealed that in each of incorrect links the Bug ID was adjacent to other alphanumeric characters forming a Bug ID with more digits or a different

Table II. Analyzing Links Obtained With Simple Bug_ID Search

Equal Links	443	74.1%
Bugs with one link	196	32.8%
Bugs with multiple links	247	41.3%
Different Links	147	24.6%
Incorrect links	120	20.1%
Potentially correct links	27	4.5%
Links From Different Repository	8	1.3%

identification code. This led us to the improved search criteria which will predetermine the bug ID surrounding characters so we constructed the following regular expression:

$$'(. * [0 - 9]|\^)' + bug_id + '(\W|\^|r|\$)' \quad (1)$$

The regular expression given in (1) defines:

- the preceding character to be any non digit character, including the start of string
- the number to be the exact match of the Bug ID we are looking for
- the following character to be any non alphanumerical character, including the end of string

In order to evaluate the performance of linking bugs and commits using regular expression given in (1), we performed the **analyses 2** and **3**. Unlike the HTTPD project, we managed to find the equal source code history for the Opennlp project even though we take it from different repository. Therefore, the linking is done upon the same input and it offers a good comparison basis. The correct links given by the benchmark dataset are compared with the links obtained by ReLink and with the links obtained by our regular expression search. The details are given in table I.

The results obtained from the **analysis 2** upon the HTTPD project indicates the regular search can link approximately the same number of bugs as the ReLink tool. The difference between these two approaches is due to the different source code inputs. Analyzing the links made by the ReLink tool and not by the regular expression search reveals that all of their links do have a bug ID number in the commit message. We looked for these commits in the GIT repository but did not find them. This means that the regular expression search would have performed just as good as the ReLink does if it had the same source code input.

In order to prove this theory, the **analysis 3** is done upon the Opennlp project comparing the results also to the benchmark dataset. The results show that our regular expression search approach actually outperforms the ReLink tool. We found all the correct links given by the benchmark and an incorrect one. The incorrectly linked commit is given in table III. The commit message clearly shows it has nothing to do with the Bud ID we are looking for. Furthermore, the time elapsed between closing the bug and committing the supposed fix is greater than it usually is (from several days up to a month). Finally, the bug assignee does not correspond to the author of the commit. This link was made due to a single digit bug ID which is found in the name of Apache release. The bug IDs that are small in value present an obstacle to our search approach because they could be found in other identification numbers, dates, release tags or similar parts of a commit message.

The ReLink, on the other hand had no incorrect links, but managed to miss 12 of them, which is approximately 10%. They also did not link 5 bugs at all, which is approximately 6%. Several examples of missing links are given in table IV. It is unclear why the ReLink tool did not make that connection. All the presented missing links do contain the Bug ID in the commit message. The time elapsed

Table III. Commit Incorrectly Linked by the Regular Expression Search

Bug ID	Commit Message	Opened	Closed	Committed	Bug Assignee	Committer
9	OPENNLP-190 Updated to Apache 9 parent pom and removed special version which we needed for the Apache 8 parent pom, namely for the rat plugin and the release plugin.	9.12.2010	13.1.2011	30.5.2011	William Colen	Joern Kottmann

between closing the bug and committing the supposed fix ranges from 0 up to 63. Besides the first commit linked to the Bug ID 367 which has 63 days and the commit linked to the Bug ID 471 which has 35 days, the remaining commits are within usual time interval. Finally, all the commit authors do correspond to the bug assignee role in the presented missing links.

Table IV. Examples of Links Missed by the ReLink Tool

Bug ID	Commit Message	Opened	Closed	Committed	Bug Assignee	Committer
84	OPENNLP-84 Corrected method name to sentPosDetect	25.1.2011	25.1.2011	25.1.2011	Joern Kottmann	joern
115	OPENNLP-115 Charset should be specified before creating input stream	1.2.2011	11.7.2011	11.7.2011	Joern Kottmann	joern
471	OPENNLP-471: found after we find a name match, we don't jump over the found name but re-process... thanks William for pointing this out	14.3.2012	24.4.2012	19.3.2012	James Kosin	jkosin

The mistakes of the linking process reflect upon the files we indicate as fault prone as well. Our approach pronounces the same 141 files to be fault prone as the benchmark dataset does, because the incorrect link is connected to the already fault prone file. The ReLink failed to pronounce 9 different files as being fault prone, which is approximately 6%.

4. CONCLUSION

Integrated datasets such as Bug-Code fix datasets are very important for further development of the software engineering discipline. This datasets can provide us new insights into the software development processes and practices and the conclusions may lead to new developments. The creation of as much as possible Bug-code datasets that are reliable and comparable enough to minimize external threats to validity is an important task. However, addressing the data quality and dataset bias is not an easy task.

In this paper we show that already a simple traditional approaches with help of regular expressions may work well. However, they may not be equally effective in all environments. One should adapt the regular expression to each particular repository and even to each project. From the performed study we may observe that the data quality issues may be lower for bigger, longer and more complex projects. Therefore, we assume that very advanced linking techniques involving learning rules might not be not only ineffective but also not applicable for the complex project environments. The very good results obtained by our regular expression bug-code linking, evident in the case of benchmark datasets, encourage us to expand this research. Our future work is to replicate it on more complex projects from different open source communities and to expand this the selection bug-code linking techniques. The comparison of techniques will include the datasets provided by other researchers and try to find statistical evidence in favor or against these statements.

The main contribution of this study is the presentation of an experiment and the preparation for more advanced and more demanding experiment that would lead to development of algorithm adaptable to open source repositories and projects aiming to build as much as reliable bug–code datasets.

REFERENCES

- Adrian Bachmann, Christian Bird, Foyzur Rahman, Premkumar Devanbu, and Abraham Bernstein. 2010. The Missing Links: Bugs and Bug-fix Commits. In *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE '10)*. ACM, New York, NY, USA, 97–106. DOI: <http://dx.doi.org/10.1145/1882291.1882308>
- Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. 1999. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
- Tegawende F. Bissyande, Ferdian Thung, Shaowei Wang, David Lo, Lingxiao Jiang, and Laurent Reveillere. 2013. Empirical Evaluation of Bug Linking. In *Proceedings of CSMR '13*. IEEE Computer Society, Washington, DC, USA, 89–98. DOI: <http://dx.doi.org/10.1109/CSMR.2013.19>
- Michael Fischer, Martin Pinzger, and Harald Gall. 2003. Analyzing and Relating Bug Report Data for Feature Tracking. In *Proceedings of the 10th Working Conference on Reverse Engineering (WCRE '03)*. IEEE Computer Society, Washington, DC, USA, 90–. <http://dl.acm.org/citation.cfm?id=950792.951355>
- Tihana Galinac Grbac, Goran Mauša, and Bojana Dalbelo Bašić. 2013. Stability of Software Defect Prediction in Relation to Levels of Data Imbalance.. In *Proceedings of SQAMIA '13*. Novi Sad, Serbia, 1–10.
- Tihana Galinac Grbac and Darko Huljenic. 2014. On the probability distribution of faults in complex software systems. *Information and Software Technology* 0 (2014), –. DOI: <http://dx.doi.org/10.1016/j.infsof.2014.06.014>
- Tihana Galinac Grbac, Per Runeson, and Darko Huljenic. 2013. A Second Replicated Quantitative Analysis of Fault Distributions in Complex Software Systems. *IEEE Trans. Software Eng.* 39, 4 (2013), 462–476. <http://dblp.uni-trier.de/db/journals/tse/tse39.html#GrbacRH13>
- S. C. Kleene. 1956. Representation of Events in Nerve Nets and Finite Automata. *Automata Studies* (1956).
- Jean Petrić and Tihana Galinac Grbac. 2014. Software Structure Evolution and Relation to System Defectiveness. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (EASE '14)*. ACM, New York, NY, USA, Article 34, 10 pages. DOI: <http://dx.doi.org/10.1145/2601248.2601287>
- Adrian Schrter, Thomas Zimmermann, Rahul Premraj, and Andreas Zeller. 2006. If your bug database could talk. In *In Proceedings of the 5th International Symposium on Empirical Software Engineering, Volume II: Short Papers and Posters*. 18–20.
- Jacek Śliwerski, Thomas Zimmermann, and Andreas Zeller. 2005. When Do Changes Induce Fixes? *SIGSOFT Softw. Eng. Notes* 30, 4 (May 2005), 1–5. DOI: <http://dx.doi.org/10.1145/1082983.1083147>
- Ashish Sureka, Sangeeta Lal, and Lucky Agarwal. 2011. Applying Fellegi-Sunter (FS) Model for Traceability Link Recovery between Bug Databases and Version Archives.. In *APSEC*, Tran Dan Thu and Karl R. P. H. Leung (Eds.). IEEE, 146–153. <http://dblp.uni-trier.de/db/conf/apsec/apsec2011.html#SurekaLA11>
- Ken Thompson. 1968. Programming Techniques: Regular Expression Search Algorithm. *Commun. ACM* 11, 6 (June 1968), 419–422. DOI: <http://dx.doi.org/10.1145/363347.363387>
- Davor Čubranić and Gail C. Murphy. 2003. Hipikat: Recommending Pertinent Software Development Artifacts. In *Proceedings of the 25th International Conference on Software Engineering (ICSE '03)*. IEEE Computer Society, Washington, DC, USA, 408–418. <http://dl.acm.org/citation.cfm?id=776816.776866>
- Rongxin Wu, Hongyu Zhang, Sunghun Kim, and Shing-Chi Cheung. 2011. ReLink: Recovering Links Between Bugs and Changes. In *Proceedings of ESEC/FSE '11*. ACM, New York, NY, USA, 15–25. DOI: <http://dx.doi.org/10.1145/2025113.2025120>

Processing and Data Collection of Program Structures in Open Source Repositories

JEAN PETRIĆ, TIHANA GALINAC GRBAC and MARIO DUBRAVAC, University of Rijeka

Software structure analysis with help of network analysis showed promising results in software engineering community. Some previous studies have presented potential of representing software structure as subgraph frequencies, that are present in software structure using network graphs, for effective program characterization and differentiation. One of the prerequisites for exploration of this potential is collecting large dataset with number of different software projects. Nowadays, there are plenty of open source code repositories which not only they contain source code, but also provide a lot of crucial information useful in guiding software engineering actions. However, systematical building of such large dataset is highly challenging and time consuming process. Therefore, our approach is to automate data collection process for open source repositories aiming to provide as large as possible dataset that could provide us reliable conclusions. In this paper we present software structure tool analyzer based on subgraph frequencies, discuss its automation opportunities and illustrate preliminary results obtained by its usage. Some potential research directions are discussed.

Categories and Subject Descriptors: H.3.0 [Information storage and retrieval] General; D.2.8 [Metrics] Product metrics

General Terms: Experimentation

Additional Key Words and Phrases: open-source repositories, automatic tool, software analysis

1. INTRODUCTION

The use of network analysis in software analysis has been widely explored recently. In [Zimmermann and Nagappan 2008] authors showed that using of network graphs metrics can provide better results in the SDP (software defect prediction) than with classical software metrics. This fact motivated our analysis of software as network graphs.

One way to approach the network graphs is to analyze its basic building blocks as subgraphs and motifs. In our previous work [Petrić and Galinac Grbac 2014] we analyzed three Eclipse systems and got some interesting preliminary results. We realized that with help of subgraph frequencies we may differentiate different software. This is an important finding because it may be useful in many software engineering tasks - from planning software design and system modeling to planning the verification activities. In fact, variations of k-node subgraph frequencies, that in sum have probability one in the analyzed graph structure, are bounded. In order to discover these bounds in software projects we wanted to empirically investigate as much as possible software structures. We are interested what are the bounds and are they related to some other software metrics. Also we want to explore these bounds across the software domains, i.e. the software written for different purposes.

Nowadays, there are plenty of open source software repositories. Source code repositories and all supporting software development repositories contain a lot of potentially useful information. However, those information are weakly explored and used for guiding and executing software projects although

Author addresses: J. Petrić, e-mail: jean.petric@riteh.hr; T. Galinac Grbac, e-mail: tihana.galinac@riteh.hr; M. Dubravac, e-mail: mario.dubravac@riteh.hr; Faculty of Engineering, Vukovarska 58, HR-51000 Rijeka, Croatia

Copyright © by the paper's authors. Copying permitted only for private and academic purposes.

In: Z. Budimac, T. Galinac Grbac (eds.): Proceedings of the 3rd Workshop on Software Quality, Analysis, Monitoring, Improvement, and Applications (SQAMIA), Lovran, Croatia, 19.-22.9.2014, published at <http://ceur-ws.org>.

posses huge potential for improving the software engineering tools and practices. On the other hand, there are serious obstacles to this problem. Firstly, data are not usually available in the form that easily generate useful information, there is a lack of systematically collected data from diverse projects that could produce sound and general theories, and development of systematic and standard data collection procedures is limited due to huge diversity of repositories data structure. For example, in the SDP area, there was a huge number of papers which did not satisfy proper collection of datasets, leading in questionable models for SDP [Hall et al. 2012]. One of the main reason was inadequately defined data collection procedure. A process of collecting proper datasets for analysis in empirical software engineering is always a tough job to do. Empirical data are occasionally exposed to bias. First problem is that many studies are done with small datasets and small number of datasets, which may represent a serious threat to validity and lead to weak generalization of conclusions. Another problem is lack of freely available data from commercial software. When some research has been done on closed software it is impossible to repeat this research on same datasets, so everything remains on trust. Third problem is a limited number of investigated software domains for same model, e.g. a model for defect prediction. Fourth problem is in validity of information collected retroactively from the project repositories. Some useful information are stored by software developers in project repository and that process may be biased. Collecting the network subgraphs is free of the data collection bias introduced by software developers. Software structure is readable directly from the source code files. A huge potential for future evolution of software engineering discipline is in development of systematic data collection procedures that would be able to collect data consistently from diverse project repositories [Wnuk and Runeson 2013]. The challenge is in developing tools and procedures that would automatically collect and explore this information over different projects and repositories. The process of reading the software structure is not dependent on the repository, therefore we think there is a potential to automatize this data collection.

In this paper we want to explore opportunities to develop a generic tool that would automatically and continuously collecting live data from huge number of software project repositories just by simple searching the web. More precisely, we want to investigate opportunities to develop a live software structure analyzer. Such tool could be of great importance to the software development community since software structure has huge effect on its reliability and evolution [Galinac Grbac et al. 2013; Galinac Grbac and Huljenic 2014; Petrić and Galinac Grbac 2014]. An another important thing which motivated us to start analyze open-source repositories and to think about an automation process is our previous work [Petrić and Galinac Grbac 2014]. Namely, we did our analysis on limited datasets which included three different Eclipse plugins with more than 30 releases in total for all three systems. A process of collecting and processing data was very long and ineffective. Thus, we have started to think about some improvements. From the above observations, a better understanding of open-source repositories structure is mandatory in sense of collecting needed knowledge for an automating process, and also for developing such tool which would be able to process retrieved data.

Two main things will be covered. Firstly, we will discuss about diversity of open-source repositories, and their properties. Some basic information about few popular repositories will be explored, like their adequacy for a generic data collection. Secondly, based on findings we will define our new developed tool for purposes of automatic collecting datasets from different repositories. The tool is modular and very easy for extending with new functions. We briefly discuss its current main functions. Finally, we observe some difficulties in this live data collection process. We succeed to collect software structures for 234 open source software projects and based on simple observations we discuss our future work and future research directions.

The paper is organized as follows - in Sect. 2 more about open-source repositories will be said, in Sect. 3 the software structure collection tool will be described in detail, in Sect. 4 we describe our

experiences of using the tool and provide some preliminary results of the collected data. Finally in Sect. 5 we will make a conclusion and say something about our future work.

2. OPEN-SOURCE REPOSITORIES

This section will describe how we approached to the open-source repository analysis, and how we determined for which repositories an automatic collecting process of data should be done. Open-source repositories are collection of data which are related to some project. Most often, such repositories are used for storing source code files, but this is not their only purpose. Sometimes, open-source repositories may contain vast number of data, which if they are processed in a proper way, may give crucial information about particular project. Today, many open-source repositories are available, and differences between them are properties they offer. For example, some repositories offer possibility for having a bug report system, some of them offer an unlimited number of projects, etc. In this research we have included only few repositories which show best overall properties, in terms of information they offer, a number of projects they have, etc. Information on many repositories over the Internet can be found in the comparative table available on Wikipedia ¹.

From the objective parameters from that table it is obvious that many repositories share same or similar properties, so we also employed subjective parameters which should help us in making an easier decision. Thus, for subjective parameters we have introduced two main categories: number of prominent projects and how easy is to implement automatic retrieval of data for those repositories. In easy to implement category we have looked only how well repositories are organized, i.e. how briefly are links to the source code files provided, and how good search filters are implemented, i.e. is it easy to separate Java projects from C++ projects. Also, this category contains only poor, average and well keywords for determine how good organization and search filters are. For each keyword we gave the grade, thus poor is equal to 0, average is equal to 1 and well is equal to 2.

From the analysis of a number of open source projects we made following conclusions:

- Not all projects contain a link to the source code repository to directly load the project, but have a link just to the repository web page from where the user should manually search for the project
- Most of the projects store the source code just into the GIT repository, or at some point during development switch to GIT repository, without providing direct link to the source code for each version of the software
- Number of projects are very old and the repository is not maintained more or is not available
- Some repositories does not allow search by project name and have not list of all projects storing the source code. Project search should be done manually by searching for specific keywords and thus are hardly to automate
- Some repositories are limited to specific communities, e.g. CodePlex is limited to the Microsoft community and contains projects that are based on their technology

After we performed analysis on several repositories, we decided that we should include GitHub, SourceForge and Eclipse repositories in our automation process, because they got best overall grades according to objective and subjective parameters.

3. SOFTWARE STRUCTURE COLLECTION TOOL

To face some of the problems we discussed in previous sections, we decided to implement a modular tool which will be able to automate some processes in data retrieval. Because we have found that some

¹http://en.wikipedia.org/wiki/Comparison_of_open-source_software_hosting_facilities

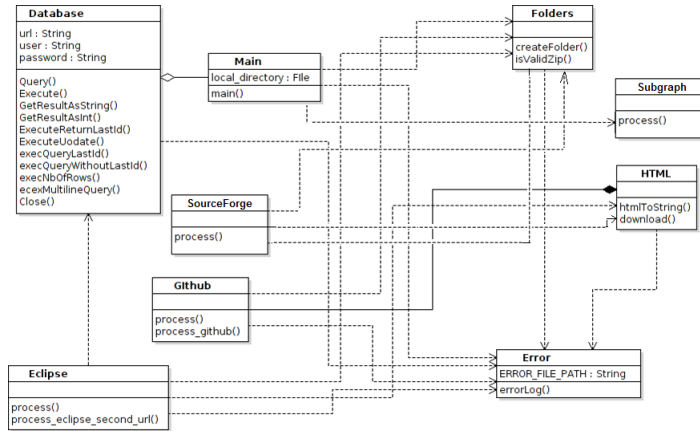


Fig. 1. Class diagram of the software structure collection tool

repositories have a good structure, i.e. a structure which is easy to be processed by some script tools, logical move was making a single tool which will be able to collect all needed data by themselves, and also to perform some additional tasks which will save time for further analysis. In that case, there is no need for manual collecting datasets from repositories, which is an important thing, because except that manual collecting data is a very time consuming process, it is also the process in which human error may have a significant impact. An automatic tool should collect all data in the same manner, which excludes occurrence of mistakes. Except aforementioned, such automatic tool is able to collect vast number of projects, so further analysis can be done on bigger datasets. Also, many software from different domains can be analyzed.

The class diagram of the implemented software structure collection tool is given on Figure 1. The software structure collection tool was mainly written in Java, but it also contains few external tools, which are rFind and JHawk. The rFind is a tool for searching relations between classes in a Java code, which is our previous work for transforming object-oriented software code into its graph representation described in [Petrić and Galinac Grbac 2014]. The software structure collection tool has few purposes, which can be divided in next phases:

- Phase 1:** automatic retrieval of a Java source code from repositories
- Phase 2:** uncompression of source code files
- Phase 3:** transformation of source code files in its graph representation with the help of the rFind tool
- Phase 4:** count of subgraphs occurrence from software representation of a graph
- Phase 5:** a software metric collection

3.1 Phase 1: automatic data retrieval

Automatic retrieval was at first the main purpose of the software structure collection tool. After the analysis we had initially chose two repositories which satisfied our criteria. Additionally, we added and third repository which was the Eclipse. Finally, we also included a built-in option in the tool. The built-in option of the tool is able to get the list of the links as an input, which leads to different source code files. After the list is provided, the tool only process those source code files. Because the process

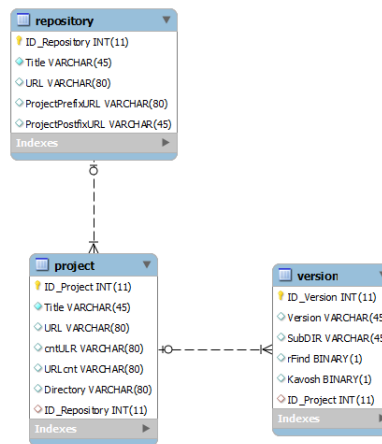


Fig. 2. An entity-relationship diagram for the software structure collection tool

can be interrupted by some unwanted events, e.g. a loss of power, we provided a recovery. The recovery enables us to be sure that already processed files will not be retrieved and processed again.

Because the software structure collection tool downloads many different types of software we have also built database for storing as much as possible information about retrieved software. The entity-relationship diagram of a database is given on Figure 2. The database contains important information about every project, such as a URL of the project, a version, etc. After each project is downloaded, the software structure collection tool continues on the phase 2. The phase 1 is repeated every time after the last phase is finished, and until there are non-processed projects in the repository.

3.2 Phase 2: uncompression of files

Immediately after the phase 1 is finished, uncompression of files begin. Repositories occasionally compress projects to preserve space on their servers, so before we can continue to the next phase, we must handle this step. Each repository uses different type of compressing files, but in most cases there are a limited number of used formats. In the software structure collection tool we have implemented support for uncompressing four different formats, which includes: zip, gz, bz2 and tar. We found that those formats are most occurring ones. If some different format is used for a compression, than this project will be discarded for the further analysis.

3.3 Phase 3: getting graph representation

This phase starts an external tool when is appropriate. After the phase 2 is finished, the rFind is called from the software structure collection tool. As is mentioned before, the rFind transforms a source code into its graph representation. To do this, it parses a code line by line and seeks for relations between classes. In that case, Java classes are nodes, and communication links are edges. A communication link means any relation between two classes. For example, if some class A tries to send a message to the class B through some method, then the edge is directed from the class A to the class B. For better understanding, the process is shown on Figure 3. Similar works also if one class tries to instantiate another class, or if it tries to communicate through parameters or a return value. For the current version of rFind, the communication in terms of class inheritance is not covered, so we do not record such relation. The rFind generates two files which are subject for further analysis. Those files are *graph* and *classlist*. Graph files contain information about communication links between classes, presented

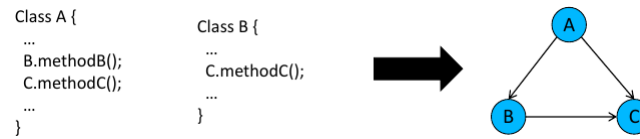


Fig. 3. Example of graph generated by rFind for given code

with IDs of the classes. To map those IDs with the corresponding classes we have a classlist in key-value format, where the key is an ID and the value is a class name.

3.4 Phase 4: subgraph frequency

The fourth phase of the software structure collection tool is a subgraph frequency counter. Its purpose is to count occurrences of all 3-node subgraphs in the given graph. We did not find any separated implementation of the subgraph counter, so we decided to use existing libraries, and adjust them to work in manner we expected. For this purpose we found the S-Space Package, which is a collection of algorithms for building Semantic Spaces as well as a highly-scalable library for designing new distributional semantics algorithms². The S-Space has already implemented algorithms for processing subgraphs, but without a subgraph counter. Thus, because the S-Space is an open-source, we used few its algorithms to implement an expected behavior.

Non-separated versions of the subgraph counter can be found in motif tools. For example, mFinder³ is a tool which provides getting of all n-node subgraphs for the given graph. The main problem is that this tool primarily searches for motifs, which is a very time-consuming process. So, our implementation of the subgraph counter has accelerated this process.

Another important thing is that our implementation of the subgraph counter can process any subgraph size, and any types of the subgraph, as long as it has correct formatted list of the subgraphs provided to the input. This means that our implementation is highly flexible. For example, if we want to find subgraph frequencies of subgraphs 1, 2 and 3 shown on Figure 4 we need to put this list as an input to the software structure collection tool:

```
1: 1-2, 1-3
2: 1-2, 3-1
3: 1-2, 1-3, 3-1
```

According to the list above, we can put any type of the subgraph and count them in the given graph. Many motif tools does not have similar option.

3.5 Phase 5: software metrics collection

Currently, the last phase in the software structure collection tool is a software metric collection. For this purpose we used the external commercial tool JHawk⁴. JHawk is a static code analysis tool, i.e. it takes the source code of your project and calculates metrics based on numerous aspects of the code, e.g. volume, complexity, relationships between class and packages and relationships within classes and packages. After this process is finished all data are stored, and the process continues with the first phase until all projects are processed.

²<https://github.com/fozziethebeat/S-Space/>

³<http://www.weizmann.ac.il/mcb/UriAlon/NetworkMotifsSW/mfinder/MfinderManual.pdf>

⁴<http://www.virtualmachinery.com/jhawkprod.htm>

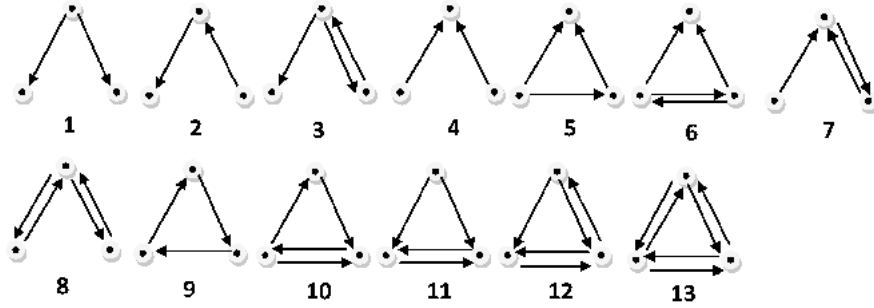


Fig. 4. All 3-node subgraphs

Table I. Descriptive statistics of the dataset

	No of 3-node subgr.	No of subgr. 6	No of subgr. 12	No of subgr. 14	No of subgr. 36	No of subgr. 38	No of packages	No of classes	No of methods	No of lines of code
Mean	80318	1593	368	0,02	78449	34	0,02	1703	10979	112511
Stdev	374760	3296	872	0,14	374529	77	0,13	2865	19737	210433
Min	3	3	0	0	0	0	1	4	66	818
Max	3291827	21343	6460	1	3287902	478	1646	16234	121811	1445451

4. EVALUATION OF EXPERIENCES AND PRELIMINARY RESULTS

In this section we discuss benefits and limitations of using the tool for the purpose of automatic data collection. Furthermore, we present some preliminary results obtained by using the tool.

4.1 Experiences of using the tool

We start data collection with help of the tool described in section 3. The total time needed for collecting data varied for different phases implemented in tool, but also from the project to project. For example, the first phase of data collection highly depends on the project and size of the source code that may vary from several MB to 200 MB. Also, data collection time in the first phase is very much depended on the Internet link throughput available to server running the data collection tool. Subsequent phases, from two to four were executed relatively fast (in few minutes) and are not so sensitive on the project size. The last phase may be very time consuming, from several minutes for smaller projects to few hours for larger ones.

In the last phase we use JHawk tool for collecting metrics on the source code files. In most cases JHawk tool performs excellent, but for some projects we experienced some problems that block our data collection process. This happens for small and large projects and we could not eliminate that issue because the JHawk is a commercial tool and we do not have insight in it. In such cases we skipped that project and continue with the next one.

4.2 Descriptive statistics for datasets collected

As a result of data collection process we succeeded to collect data for 233 projects. For each project we counted subgraph frequencies for all 3–node subgraphs and collected all metrics on the class and system level (provided by JHawk tool). Descriptive statistics for the collected projects are given in Table I.

Figure 5 depicts the relative subgraph frequencies of 3–node subgraphs (6, 12, and 36) that are present in all analyzed projects. We did not provide the figure of relative subgraph frequencies for

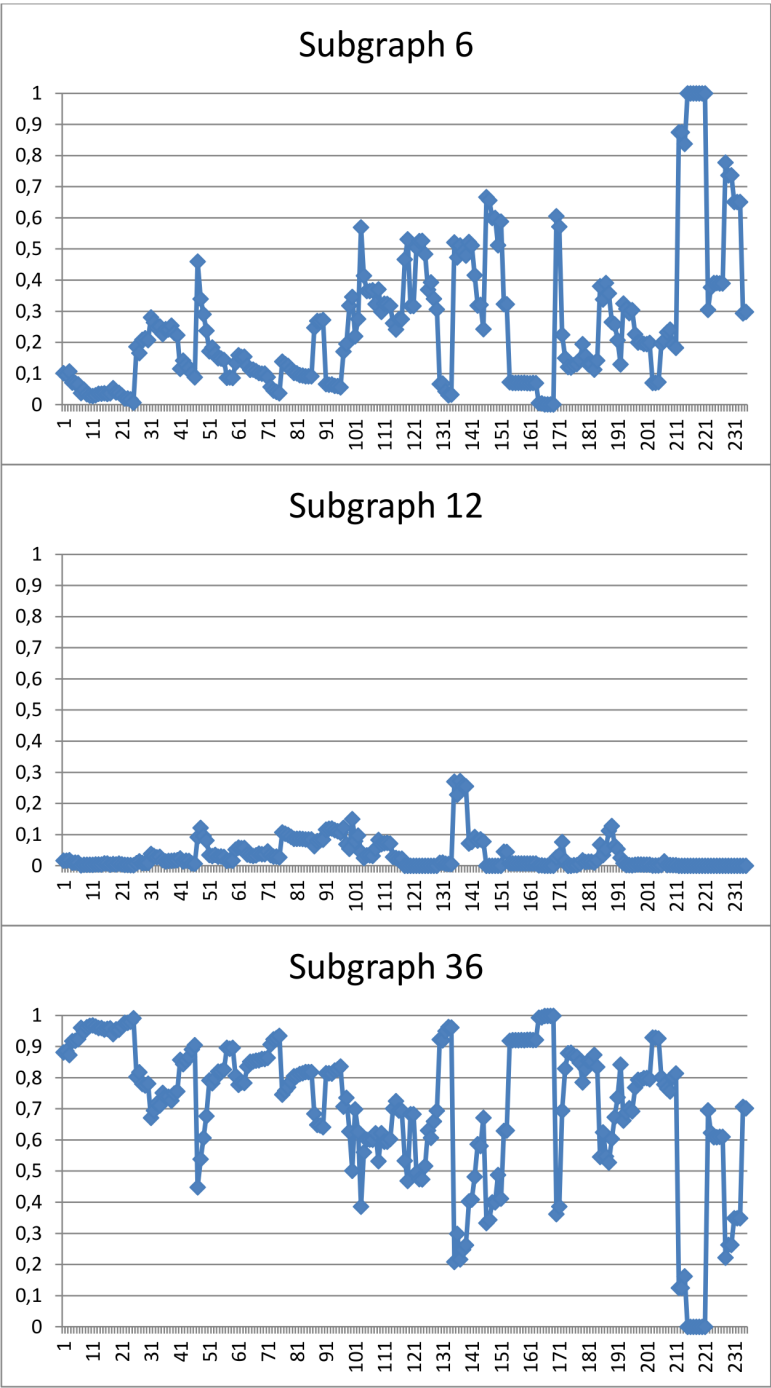


Fig. 5. Relative subgraph frequencies (3-node: 6, 12, 36) per software project. Note that software projects are ordered with respect to number of classes

Table II. Distribution of projects with respect to number of classes

No. of classes	0–200	201–400	401–600	601–800	801–1000	1001–1200	1201–1400
No. of projects	78	35	19	14	12	13	2
No. of classes	1401–1600	1601–1800	1801–2000	2001–2200	2201–2400	2401–2600	2601–2800
No. of projects	3	6	0	5	1	1	0
No. of classes	2801–3000	3001–4001	4001–5000	5001–6000	6001–7000	7001–8000	8001–9000
No. of projects	6	6	4	3	4	8	8
No. of classes	10001–11000	11001–12000	12001–13000	13001–14000	14001–15000	15001–17000	
No. of projects	0	1	1	2	1	0	

nodes 14 and 38 because are neglected, like for all other subgraphs because they are not present in neither of the analyzed projects. In all subfigures on Figure 5 the projects are ordered with respect to the number of classes present in the project source code. In the analyzed sample we have very well represented source codes with number of classes ranging from 0 to 8000. We conclude this from the distribution of projects collected over the categories with amount of classes given in Table II.

As it can be observed from the graph the frequencies of subgraph 6 are increasing with number of classes present in the project. On the other hand, the frequencies of subgraph 36 are decreasing.

We analyzed the occurrences of each 3-node subgraph (see Figure 4) and presented results in the separate subfigures for each subgraph that is represented in the analyzed projects. We found that all projects contain only subgraphs 6, 12, 14, 36, 38, 46, and 74. However, subgraphs 46 and 74 are very rarely represented in all of the analyzed projects, so we did not provide separate figure here. The most represented subgraph in all projects is the subgraph 38.

4.3 Limitations and future work

Future work should consider to improve the data collection tool with repositories that contain bigger projects. Since, the GIT repositories become a norm in the open source community the future work should consider how to automatically read structure data from all open projects stored in GIT.

Based on the observations while running the tool aiming to collect the dataset, we concluded that some phases may be computationally demanding. Future work should consider to eliminate execution stops due to data quality (e.g. problems with JHawk) and how to run the tool within the Cloud environment and without human intervention.

We aim collecting more datasets in the future that would cover wider spectra of projects than it is currently covered by the collected dataset. Some conclusions are limited in generalization due to relatively small projects that are collected. In our initial dataset mean subgraph frequencies of all analysed projects is 80k and 112 kLOC and for example some Eclipse projects that we collected have more than million 3-node subgraphs and about million lines of code. From this initial analysis we observed that the probability of particular subgraph occupancy in software may depend on the amount of classes, more packages, and more lines of code. We want to further study this observation and we

are interested how we can define and develop models that would be useful for designing and verifying software systems.

5. CONCLUSION AND FURTHER WORK

In this paper we have presented a basic analysis of software structure that is collected from the open-source repositories and we have also introduced our modular software structure data collection tool.

With the software structure collection tool we have remarkably boosted our process of collecting datasets. Except that, we have also achieved an another important thing, which is avoiding of human errors in the collection process. Moreover, we identified weaknesses of our tool and discuss its future extensions.

From the collected datasets we performed a preliminary analysis and discuss future research directions in data collection. Such automation process will help us in future to collect a significant number of projects, that could truly represent a software from different domains on which we are going to perform additional analysis in terms of the subgraph frequencies and software metrics as continuation of our previous work [Petrić and Galinac Grbac 2014].

REFERENCES

- Tihana Galinac Grbac and Darko Huljenic. 2014. On the Probability Distribution of Faults in Complex Software Systems. *Information and Software Technology* (2014). DOI: <http://dx.doi.org/10.1016/j.infsof.2014.06.014>
- Tihana Galinac Grbac, Per Runeson, and Darko Huljenic. 2013. A Second Replicated Quantitative Analysis of Fault Distributions in Complex Software Systems. *Software Engineering, IEEE Transactions on* 39, 4 (April 2013), 462–476. DOI: <http://dx.doi.org/10.1109/TSE.2012.46>
- Tracy Hall, Sarah Beecham, David Bowes, David Gray, and Steve Counsell. 2012. A systematic literature review on fault prediction performance in software engineering. *Software Engineering, IEEE Transactions on* 38, 6 (2012), 1276–1304.
- Jean Petrić and Tihana Galinac Grbac. 2014. Software structure evolution and relation to system defectiveness. In *EASE 2014 (May 13–14, 2014)*. ACM Digital Library, 10. DOI: <http://dx.doi.org/10.1145/2601248.2601287>
- Krzysztof Wnuk and Per Runeson. 2013. Engineering Open Innovation a Framework for Fostering Open Innovation. (2013). http://dx.doi.org/10.1007/978-3-642-39336-5_6
- Thomas Zimmermann and Nachiappan Nagappan. 2008. Predicting Defects Using Network Analysis on Dependency Graphs. In *Proceedings of the 30th International Conference on Software Engineering (ICSE '08)*. ACM, New York, NY, USA, 531–540. DOI: <http://dx.doi.org/10.1145/1368088.1368161>

Tool for Testing Bad Student Programs

IVAN PRIBELA, DONI PRACNER and ZORAN BUDIMAC, University of Novi Sad

This paper presents an effort to address efficient assessment of less than perfect students' solutions in a semi-automated code assessment process. Automated and semi-automated code assessment has its drawbacks when it comes to the poorly written and often non compiling programs created by beginner students. The usual assessment automation approaches often lead to adaptation of courses and assignments to the assessment process, while we try to avoid such trend and thus improve the students' experience. The solution proposed in this paper focuses on automation of the assessment process itself as opposed to the automation of grading which is the usual approach. This paper presents the concept and design of a tool aimed to support such process as well as presents two case studies that validate the use of the proposed tool.

Categories and Subject Descriptors: K.3.2 [**Computers And Education**]: Computer and Information Science Education—*Computer science education*; D.2.5 [**Software Engineering**]: Testing and Debugging—*Testing tools*

General Terms: Management, Measurement, Verification

Additional Key Words and Phrases: code assessment, manual assessment, semi-automated assessment, test-based assessment

1. INTRODUCTION

Nowadays, teachers, especially on computer science classes, are faced with grading an ever increasing number of students' assignments. There are three ways to avoid overburdening them: reducing the number of students, increasing the number of teachers, and transferring some of the load from the teachers to the computers. As it is obvious that the first two approaches are often not feasible, getting help from computers and introducing some form of automatic assessment in everyday practice becomes the only possible way of dealing with this issue in an efficient way. This approach, at least theoretically, also increases objectivity of the grading process.

However, fully automated code assessment, with all the benefits it brings, also has its drawbacks. The major one lies in the fact that not everything can be tested by a machine. Typical examples are finer points of coding style like the correct use of procedures and recursion, which are very hard to catch even by very complex metrics. Unfortunately, many approaches in automated assessment are focusing on automation of grading of all aspects of students' solutions.

This often lead to adaptation of courses and assignments to automated assessment, while the opposite should be preferred. Such trend is inappropriate and can prove to be very unfavorable to students, especially on first year programming courses.

The main problem arises from the fact that the beginner students are still learning how to program. They are not yet knowledgeable and disciplined enough to follow strict and rigid program specifications

This work is partially supported by Ministry of Education and Science of the Republic of Serbia, through project no. III 47003: Infrastructure for technology enhanced learning in Serbia;

Author's address: Ivan Pribela, Doni Pracner, Zoran Budimac, Department of Mathematics and Informatics, Faculty of Sciences, University of Novi Sad, Trg Dositeja Obradovića 4, 21000 Novi Sad, Serbia; email: pribela@dmf.rs doni.pracner@dmf.rs zjb@dmf.rs

Copyright © by the paper's authors. Copying permitted only for private and academic purposes.

In: Z. Budimac, T. Galinac Grbac (eds.): Proceedings of the 3rd Workshop on Software Quality, Analysis, Monitoring, Improvement, and Applications (SQAMIA), Lovran, Croatia, 19.-22.9.2014, published at <http://ceur-ws.org>.

that are often required by automated testing systems. These students often struggle while creating even simple programming solutions that barely compile and execute.

Such strict rules for the program format and output are perfectly acceptable for experienced students or applicable in coding competitions. However, it is very hard to use them with beginners within their introductory programming classes or generally with less successful students that have difficulties coping with the basics of compilation and program execution.

This results in a rather harsh treatment of nearly satisfactory solution, where the students can lose unproportionally large amount of points to a trifle error. This can be called extreme objectivity, and is easily removed by manual inspection of the student solution.

We propose a better approach which adapts the automated testing to first year courses, and not vice versa. More precisely our proposal is a semi-automated approach that automates the assessment process itself and leaves the finer points like the grading to the instructor.

The rest of the paper is organized as follows. The second section elaborates on the need of a tool that would support the proposed approach. The third section presents one solution that tries to tackle the described problem while the following section offers two typical example situations that illustrate the applicability of the proposed tool in a real learning environment. Finally, a brief conclusion and plans for further work are given in the final section.

2. RELATED WORK

Automatic and semi-automatic assessment of student programming assignments started as early as 1960. Among the first authors were Hollingsworth [Hollingsworth 1960] and Wirth [Forsythe and Wirth 1965], whose systems were designed for assembler and Algol. Since then many other assessment tools were developed [von Matt 1994] that usually followed modern concepts introduced by new operating systems and new programming languages.

Unfortunately, many such ventures focus on a specific programming language or a specific platform, like the systems described in [Dempster 1998; Hawkes 1998; Joy et al. 2005] that focus on Java programming language. This is in line with the trend of Java being the one of the most widely used programming languages on introductory courses. There is also sparing support for other popular languages like Pascal, C/C++ and Python.

There were though some attempts to create a system for testing that will encompass wider spectrum of programming languages, like the one developed in Python presented in [Amelung et al. 2006]. These automated assessment systems are usually language independent if the assessment is based on comparison of the output.

There were also attempts to bring automated assessment of programming to learning management systems (LMS). The assessment system described in [Botički et al. 2008] addresses assessment of SQL Select queries and short programming assignments in a custom LMS environment, providing support for multiple programming languages. However, that solution is limited to the AHyCo LMS developed by its authors and lacks support for large programming projects.

Most contemporary efforts to address the issue of testing non compiling solutions are Web based and offer only visual inspection, like Assyst [Jackson and Usher 1997], which offers a graphical interface that can be used to direct all aspects of the grading process.

BOSS [Joy et al. 2005] is also a Web-based tool, which supports the whole of the assessment process and does not constrain the teacher to present or deliver their assessment material in any given style. Support for software metrics and for unit testing is covered as well.

Another fully functional assessment system called Testovid [Pribela et al. 2011] in comparison to the mentioned systems is built on Apache Ant and thus is not dependent on any programming lan-

guage or specific building and compilation logic. Furthermore, the system is used in a wide variety of situations and environments, as it is very extensible, modular, and can quickly adapt to new trends. It supports both fully automatic and semi-automatic assessment and can be extended to support manual assessment as well.

In their paper [Ahoniemi and Karavirta 2009], the authors Ahoniemi and Karavirta focus on the manual assessment and analyze the use of a rubrics-based grading tool on larger courses with multiple graders. Their results show that the use of such tools can support objective grading with high-quality feedback with reasonable time usage. They also give some pointers for teachers intending to adopt such tools on their courses.

Auvinen goes a step further in his tool called Rubyrac [Auvinen 2011] and presents a tool for rubric-based assessment that helps course staff construct textual feedback for students. The tool allows teachers to create grading templates that specify the evaluation criteria and contain feedback phrases for typical mistakes. Because many students make similar mistakes, large portions of feedback can be constructed from pre-written blocks making it possible to give detailed feedback and thus automate the process of grading with little effort.

Another effort at automating the grading process while keeping the human at the center is described in [MacWilliam and Malan 2013]. On a CS50 introductory course at Harvard University, when students complete programming assignments they have traditionally received feedback from staff in the form of comments via email. Staff reported spending significant amounts of time grading because of bottlenecks that included generating PDF documents and manually emailing feedback to students. As it is always preferable that the staff spend less time on logistics and more time providing feedback and helping students, the authors of the paper set out to improve the efficiency of the grading process by creating a web-based utility through which staff can leave feedback for students. This resulted in a 10% fewer hours per week and 13% fewer minutes per student, even while providing as much or more feedback.

Another similar approach with assessment [DeNero and Martinis 2014] aimed to improve the composition quality of student programs argues that the program structure can be understood effectively only by a person. In their paper authors describe their experiences on manually grading over 700 students by a staff of only 10 human graders. To facilitate this effort, they created an online tool that allows teachers to provide feedback efficiently at scale.

However, another topic to keep in mind with manual computer-based assessment for handling large-scale courses is the assessment granularity. The forms that the assessment takes can vary widely from simple acknowledgement to a detailed analysis of output, structure and code. The study [Falkner et al. 2014] analyses the degree to which changes in feedback influence student marks and persistence in submission. In their work, the authors collected data for over a four year period and for over 22 courses. They discovered that pre-deadline results improved as the number of feedback units increase and that post-deadline activity was also improved as more feedback units were available. Unfortunately, greater granularity also means more load for the teachers thus a balance must be found.

3. THE AUTOMATION SCRIPTS

Based on our own problems and experiences in grading large number of student solutions, a set of helper scripts have evolved over a period of time. These scripts have been in use with an array of different assignments in several programming courses on our Department.

The approach assumes that the student solutions are all stored together in one folder with a subfolder corresponding to each student. The naming of the subfolders and the method used to collect all the solutions, be it an automated submission system or manual collection, is of no importance. Al-

though it is recommended that the subfolders contain the student's name or id this is not required by the scripts.

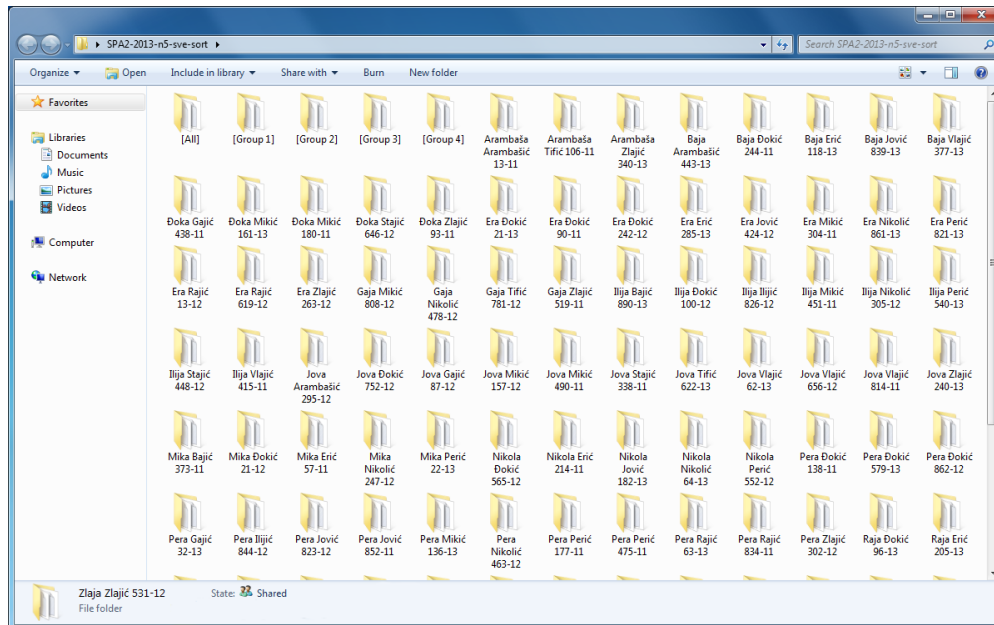


Fig. 1. Folder layout for a typical assignment

A list of the students to be graded is supplied to the scripts via a file usually named “students.txt”. This list can be generated from folder names, although most of the courses have some sort of an electronic list of the students, probably in a spreadsheet or exportable as such. Such spreadsheets can also be used for this purpose. Since it is possible that the folder names are different than the actual student names, the input can either be a list of the folders or a table with folder identifications and student names and whatever additional data is needed or wanted. In our system we use the student id, their names and groups numbers. A different scenario might also use date and time of last submission, classroom number or whatever might be relevant.

The current implementation of the system uses Apache Ant [Loughran and Hatcher 2007] as the central driver of the process. Ant is a Java library that was originally developed to help with the build process for complex Java projects, with the goal of making something similar to Makefiles for C projects, but at the same time platform independent since it would be running in the Java Virtual Machine. The two basic concepts in an Ant build file are tasks (individual actions such as compilation) and targets (more complex combinations of actions – a number of tasks in order). It provides a number of built-in tasks for compiling, testing and running Java programs, but also to work with filesystems and other types of programs. It can also be easily expanded with custom tasks, and is therefore applicable to any process that is defined in the above terms.

At the core of switching between student solutions to be graded is the custom Ant task that loads up the next solution. The data about the current student being graded is held in a file called “current.properties”. The file is in the standard Java properties format, allowing all the data to be accessible to the scripts or other tools as needed. If the file doesn't exist, the assumption is made that the grading has just begun and the first student is to be chosen. Otherwise, the student list is consulted

and the first student after the currently selected one is chosen. Either way the data from the list will be put into the “current.properties”, the content of the temporary folder will be deleted and then populated with files relevant to the next student solution.

The content of this temporary folder consists of combined files from several folders. The script first copies the content from a universal folder with files and data relevant for all students on a course. This folder usually contains basic test files that are not related to the specific group or helper libraries that are used in the assignment. Next, contents of a folder for the corresponding assignment is collected. Most often this folder contains test files for the assignment. Finally the files that represent the student’s solution are copied over to the temporary folder. This final folder stores all files the student has created for the assignment. By default, the script will copy over all the files found in this folder. This behaviour can be overridden and the set of files copied restricted by any regular expression or a set of expressions.

If needed, the list of the included folders can be easily expanded. Apart from the described folders, any files from another source can be included in the content of the temporary folder. These new sources can include supplemental library files, folders with correct solutions, or practically anything that is needed.

Besides helping with the set up of the testing area, the scripts can also help with the actual testing. This help can be in the form of program compilation, using Ant tasks defined for this, running of the student’s solution with different inputs, which may vary depending on the student’s assignment, or basically anything else supported by Apache Ant.

One of the pre-built tasks is used for execution of the student’s program with the simplest input for a quick initial test. This allows the teacher to make manual changes to the program before running the second built-in task that will run all the inputs from the temporary folder that are described by a regular expression. Most of the assignments will also have a task to run the (partial) correct solution for comparison, as well as support to open up the source code of the submitted program and the correct solution or a snippet of code in an editor or difference comparison program. This all allows for saving time on repetitive tasks while giving maximal flexibility in the process for manual interventions.

The advantage with building the system around Apache Ant scripts is that support is widespread, allowing for their usage in many different set-ups, such as Eclipse or NetBeans plugins, a lightweight editor, or even directly from the command line, possibly as part of a larger script. Additionally there are a lot of people familiar with the standard which makes customisations to the scripts much more accessible.

4. TYPICAL USE CASES

To illustrate the proposed approach and the implemented system, we will describe two typical use cases. We have selected assignments from the course Data Structures and Algorithms course on our Department that best illustrate the scripts in action.

The first usage example of the proposed scripts is an assignment that is common in the education of programming – sorting data. The assignment is to load some data from a file, use a specified sorting algorithm and save the data to another file. The students are organised into groups due to spacial limitations, so different groups will have different types of data. An item in the array to be sorted will typically be some form of a record, such as information about people, ie. name/surname/birth year, or an array, such as measurements of temperatures during a week. On this data one of the three possible elementary sorting algorithms should be applied with a simple sorting key, and some additional constraint as a bonus task (for instance sorting by a second criteria).

Automated testing on an assignment such as this proves to be rather difficult for anything other than the fully correct solution. The students often make small mistakes with the indexes in the al-

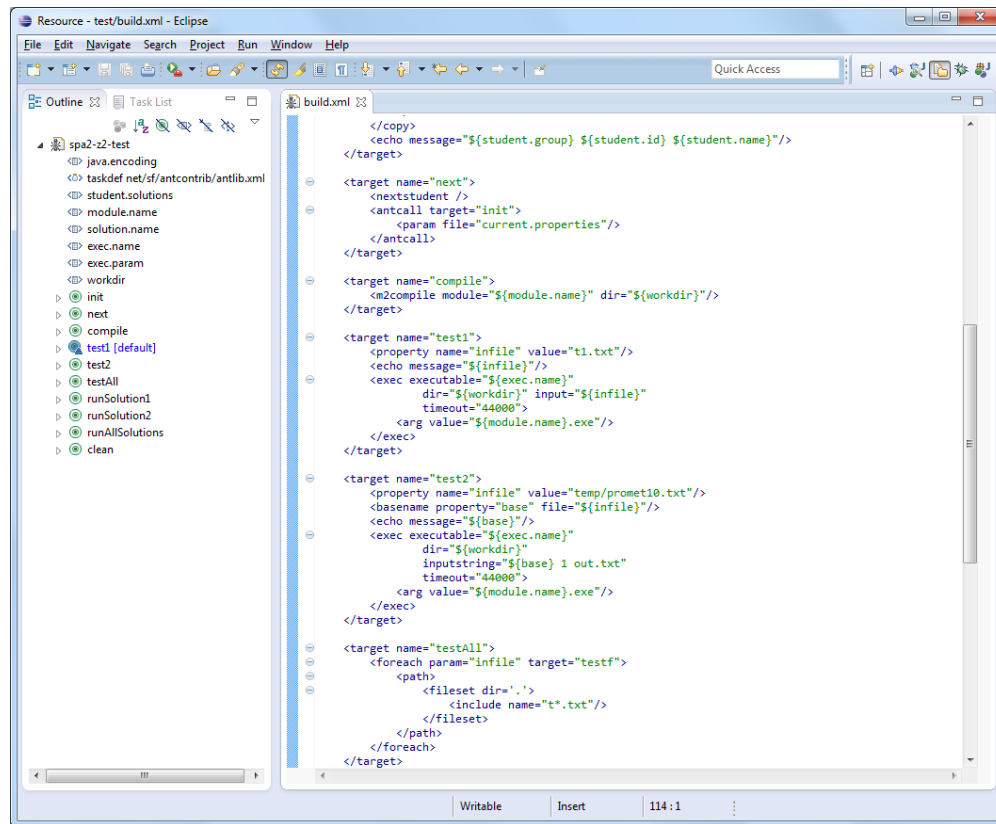


Fig. 2. Script for an assignment loaded into eclipse

gorithms which can result in partially sorted arrays, duplicated items, or even crashes and endless loops. Although these can be very close to the correct solution, an automated tester would give such a solution an unproportionally low amount of points.

Apart from this, it is hard to automatically detect whether a solution that gives the correct result was made with the correctly selected algorithm. The selection of the appropriate sorting algorithm, not just its implementations, is one of the main tasks of this assignment.

Another common problem for automated testing is that students make mistakes in saving the results in the file, forgetting to put in breaks between data or mixing up the order of the fields.

For the second use case we took another assignment common for computer science curricula – searching. Namely the illustrated algorithm is backtracking. One of the assignments that are used on the same course is to find a path through a maze presented in an ASCII text file.

The problem is easy to understand for the students, it illustrates the principles very nicely and it also allows for variance that is required to make different assignments for different groups. For instance there can be “classical” mazes with walls, a starting point and an end point. There can additionally be collectables spread through the maze (pieces of gold or chocolate, radiation levels...) with the task to either maximise or minimise the route in some way (length, collectables collected...). There can be mazes with special rules, such as using only odd numbers to move, or jump in certain directions, different heights that can only be traversed downwards, etc.

This assignment also has the problem of being hard to automatically grade with anything other than fail or pass for a number of inputs. Again there are often mistakes that lead to endless loops or recursion, typos or wrongly defined transitions, all of which are hard to algorithmically detect, and might lead to all of the outputs being wrong, while the code still shows the basic understanding of the problem at hand.

Our tool can help the instructor with these types of submissions in several ways. The testing area is being rebuild for each student from scratch with correct inputs for the student's group. Once a submission has been graded, the next student will be automatically loaded. Both of these save time and increase the reliability of the process.

The instructor can run a number of pre-defined tests on the current submission starting with a very basic test. If it fails there is no use in running the more complex ones straight away, especially if the fault is in the format of the output for instance. Instead the instructor can mark down the detected problem and make manual changes to fix them and then re-run the tests to check for further faults. Correct solutions to the problem can also be run on the same inputs for comparison.

To further help with the grading there are also options to run comparisons to pre-defined blocks of code that contain the correct solutions, either as complete programs, or just snippets that apply to a particular part of the assignment. This can help in detecting subtle mistakes such as array index problems, or special cases that are often overlooked and can pass undetected by a worn instructor. It can also help if there was a prescribed method of solving the problem – as in our sorting use case, where a particular algorithm is part of the assignment.

5. CONCLUSIONS AND FUTURE WORK

The proposed scripts can help greatly with the repetitive tasks during the manual grading. Typical assignments that show the benefits of using such scripts are illustrated in the previous chapters. The teacher only needs to monitor a single folder. The student's submissions will be copied one by one automatically without even a need to manually look up who the next student is let alone where the solution and other files are. The appropriate test cases will be loaded as well, and the grader has an array of options that can be applied as the situation demands. These options can range from an application of simple input files to an executing program, to the comparison of student code with snippets of correct solutions. Also, only one test case can be applied, all of the prepared test cases, or an ad hoc example. All the time during such grading the teachers have the options for manual changes of the students' code as needed and can combine all the mentioned options as they like.

The proposed scripts greatly decrease the time needed to set up and tear down a testing environment for fair grading of solutions. At the same time they also eliminate a lot of the possible sources of mistakes that are common for repetitive tasks.

Our current research efforts are directed at gathering more data on the impact of this approach on the time teachers spend on assessment compared to standard two-phase approach. The experiences in using these scripts for the last few years have been positive.

Another objective is to collect more usage examples and then improve the scripts to cover more scenarios during the manual testing or incorporate the scripts in a larger environment. This is aimed to maximize the help offered to the teachers and further shorten the time needed for the repetitive tasks.

ACKNOWLEDGMENTS

The work is partially supported by Ministry of Education and Science of the Republic of Serbia, through project no. III 47003: Infrastructure for technology enhanced learning in Serbia.

REFERENCES

- Tuukka Ahoniemi and Ville Karavirta. 2009. Analyzing the Use of a Rubric-based Grading Tool. *SIGCSE Bull.* 41, 3 (July 2009), 333–337. DOI: <http://dx.doi.org/10.1145/1595496.1562977>
- Mario Amelung, Michael Piotrowski, and Dietmar Rösner. 2006. EduComponents: Experiences in e-Assessment in Computer Science Education. *SIGCSE Bull.* 38, 3 (June 2006), 88–92. DOI: <http://dx.doi.org/10.1145/1140123.1140150>
- Tapio Auvinen. 2011. Rubyric. In *Proceedings of the 11th Koli Calling International Conference on Computing Education Research (Koli Calling '11)*. ACM, New York, NY, USA, 102–106. DOI: <http://dx.doi.org/10.1145/2094131.2094152>
- Ivica Botički, Ivan Budišćak, and Nataša Hoić-Božić. 2008. Module for online assessment in AHyCo learning management system. *Novi Sad J. Math* 38, 2 (2008), 115–131.
- Jay Dempster. 1998. Web-based assessment software: Fit for purpose or squeeze to fit. In *Proc. of 2nd Computer Assisted Assessment Conference*.
- John DeNero and Stephen Martinis. 2014. Teaching Composition Quality at Scale: Human Judgment in the Age of Autograders. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education (SIGCSE '14)*. ACM, New York, NY, USA, 421–426. DOI: <http://dx.doi.org/10.1145/2538862.2538976>
- Nickolas Falkner, Rebecca Vivian, David Piper, and Katrina Falkner. 2014. Increasing the Effectiveness of Automated Assessment by Increasing Marking Granularity and Feedback Units. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education (SIGCSE '14)*. ACM, New York, NY, USA, 9–14. DOI: <http://dx.doi.org/10.1145/2538862.2538896>
- George E. Forsythe and Niklaus Wirth. 1965. *Automatic Grading Programs*. Technical Report. Stanford University, Stanford, CA, USA.
- Trevor Hawkes. 1998. An experiment in computer-assisted assessment. In *Proc. of 2nd Computer Assisted Assessment Conference*.
- Jack Hollingsworth. 1960. Automatic Graders for Programming Classes. *Commun. ACM* 3, 10 (Oct. 1960), 528–529. DOI: <http://dx.doi.org/10.1145/367415.367422>
- David Jackson and Michelle Usher. 1997. Grading Student Programs Using ASSYST. *SIGCSE Bull.* 29, 1 (March 1997), 335–339. DOI: <http://dx.doi.org/10.1145/268085.268210>
- Mike Joy, Nathan Griffiths, and Russell Boyatt. 2005. The Boss Online Submission and Assessment System. *J. Educ. Resour. Comput.* 5, 3, Article 2 (Sept. 2005). DOI: <http://dx.doi.org/10.1145/1163405.1163407>
- Steve Loughran and Erik Hatcher. 2007. *Ant in Action* (second ed.). Manning Publications. 600 pages. <http://antbook.org/>
- Tommy MacWilliam and David J. Malan. 2013. Streamlining Grading Toward Better Feedback. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '13)*. ACM, New York, NY, USA, 147–152. DOI: <http://dx.doi.org/10.1145/2462476.2462506>
- Ivan Pribela, Mirjana Ivanović, and Zoran Budimac. 2011. System for Testing Different Kinds of Students' Programming Assignments. In *Proceedings of 5th International Conference on Information Technology ICIT*.
- Urs von Matt. 1994. Kassandra: The Automatic Grading System. *SIGCUE Outlook* 22, 1 (Jan. 1994), 26–40. DOI: <http://dx.doi.org/10.1145/182107.182101>

Toward Language Independent Worst-Case Execution Time Calculation

GORDANA RAKIĆ and ZORAN BUDIMAC, Faculty of Science, University of Novi Sad

Set of Software Quality Static Analyzers (SSQSA) is a set of software tools for static analysis that is incorporated in the framework developed to target the common aim – consistent software quality analysis. The main characteristic of all integrated tools is the independency of the input computer language. Language independency is achieved by enriched Concrete Syntax Tree (eCST) that is used as an intermediate representation of the source code. This characteristic gives the tools more generality comparing to the other similar static analyzers. The aim of this paper is to describe an early idea for introducing support for static timing analysis and Worst Case Execution Time (WCET) calculation at code level in SSQSA framework.

Categories and Subject Descriptors: **D.2.8 [Software engineering]: Metrics - Performance measures**

General Terms: Languages, Experimentation, Measurement

Additional Key Words and Phrases: Worst case execution time, language independency

1. INTRODUCTION

The quality of each product, and therefore also the quality of the software product, can be described as the degree to which a given product meets the needs and requirements of users. Software quality model defined by standard ISO 9126-1¹ distinguishes six attributes of software quality: functionality, usability, reliability, efficiency, portability, and maintainability.

The mentioned attributes of software quality can be monitored, evaluated, and controlled at early stages of software development by examining the source code and other static artefacts, or during the execution and testing process. Assessment of software quality attributes that is made on the source code or any of its internal representations without executing the program is called static analysis, while analysis of the program during execution time is called dynamic analysis. In the modern approach of software development, a great importance is given to monitoring and quality control in the early stages of development. Therefore static analysis becomes more important.

One of the important quality attributes of real-time systems is the execution time. It is highly important for these systems to provide required services on time. One of the parameters to be measured in order to guarantee this attribute in real-time system quality monitoring is Worst Case Execution Time (WCET) [Wilhelm et al. 2008; Lokuciejewski and Marwedel 2009; Lokuciejewski and Marwedel 2011]. It is measured as a part of timing analysis and provides value of the longest execution time of a program that can ever occur. It can also be predicted as a part of static analysis as well as measured as a part of dynamic analysis.

This paper provides early research toward support of static timing analysis and WCET calculation in SSQSA framework. In doing that, we shall take ALF as a domain specific language for WCET [Gustafsson et al. 2009] as our starting point and import the basic timing data into a SSQSA framework. The main difference between SSQSA set of static analyzers and ALF is in their level of abstraction: while ALF is intermediate *language*, enriched Concrete Syntax Tree - eCST (on which SSQSA framework is based) is a universal intermediate *data structure*. Furthermore, there are only several translators to ALF (C, C++,

This work was partially supported by the Serbian Ministry of Education, Science and Technological Development through project "Intelligent Techniques and Their Integration into Wide-Spectrum Decision Support", no. OI174023 and by EU ICT COST action IC1202 "Timing Analysis on Code-Level (TACLe)".

Author's address: G. Rakić, Z. Budimac, Department of Mathematics and Informatics, Faculty of Sciences, University of Novi Sad, Trg Dositeja Obradovića 4, 21000 Novi Sad, Serbia, email: {goca, zjb}@dmi.uns.ac.rs.

Copyright © by the paper's authors. Copying permitted only for private and academic purposes.

In: Z. Budimac, T. Galinac Grbac (eds.): Proceedings of the 3rd Workshop on Software Quality, Analysis, Monitoring, Improvement, and Applications (SQAMIA), Lovran, Croatia, 19.-22.9.2014, published at <http://ceur-ws.org>.

¹ <http://www.sqa.net/iso9126.html>

some assembler languages)² [Gustafsson et al. 2009] while SSQSA supports many more high-level languages (Modula-2, Delphi, Java, C#,...). By including ALF into a SSQSA environment, we hope that we can introduce the WCET analysis to a much broader class of languages.

In the rest of the paper we describe the planned approach. Background for a described idea is provided in the subsequent section, while section number 3 illustrates the idea. Related work is provided by section number 4. Conclusion with plans for future work is given in the last section.

2. BACKGROUND

The basic idea for integrating static timing analysis into SSQSA framework is to introduce support for domain specific languages (e.g. ALF [Gustafsson et al. 2009]) and to use the eCST Generator [Rakić and Budimac 2013, Kolek et al. 2013] to generate eCST containing all needed information. Exporting of timing information is to be done by attaching timing attributes to universal nodes representing specific program constructs as proposed by [Parsa and Mehdi 2014].

2.1 SSQSA internal representations of a software product

The main characteristic of SSQSA framework is its independency of the input computer language based on language independent tree representation of the source code - enriched Concrete Syntax Tree (eCST). Basic concept used here is to use universal nodes to enrich syntax tree so that they annotate semantics of the construct in its sub-tree. Universal nodes are constructed in three levels:

- High-level eCST universal nodes mark entities declaration on the architectural level. Interface-level declarations of packages, classes, modules and methods, procedures, functions, etc. , as well as explicitly stated high-level relations between them (such as inheritance, instantiation, implementation, etc.).
- Middle-level eCST universal nodes are those used at the level of entity definition. They appear in the body of the entities and mark individual statements, groups of statements or parts of statements with appropriate concept expressed by them (jump statement, loop statement, branch statement, condition, import statement, etc.).
- Low-level eCST universal nodes are universal nodes that mark individual tokens with appropriate lexical category (keywords, separators, identifiers, etc.).

Based on the eCST representation of the source code we can (independently of an input language) generate other source code representations.

Generation of eCFG (enriched Control Flow Graph) is one of the first tasks to be completed toward the static timing analysis in the SSQSA framework. This is to be done based on a middle and low level universal nodes. Work on this task has been recently finished.

Furthermore, based on mainly high-level universal nodes we can generate different kind of software networks completely independently of an input language [Savić et al. 2014]. For timing analysis, the most important network is one corresponding to a call graph. Based on an eCFG and this network we can create inter-procedural CFG.

Based on these program representations we can implement any of widely used approach to determine upper bound of execution time (tree based, path based, implicit path enumeration, etc.) and supporting analysis [Wilhelm et al. 2008, Lokuciejewski and Marwedel 2011]. However, for implementation of the WCET calculation algorithm we need to additionally enrich eCST by timing analysis specific attributes. This is to be introduced as XML attributes added by postprocessor based on ALF representation of the source code.

2.2 ALF language integration

ALF is an intermediate language used to represent input code written in language on high, middle, or low level. It can be also used to represent intermediate codes. Code represented by ALF language is adapted so that the WCET calculation is enabled. In other words, it contains all needed information.

² <http://www.mrtc.mdh.se/projects/wcet/home.html>

Currently, three translators exist: a translator from C/C++ to ALF, translator from proprietary IAR intermediate code to ALF, and translator from binary code to ALF.

By generating eCST for source code represented by ALF code we will be able to run WCET algorithms on it.

As described in [Kolek et al. 2013], we will need a grammar for ALF language in order to produce scanners and parsers to be used in eCSTGenerator. Furthermore, we have to determine which universal nodes are needed to implement algorithm. From this point of view the most important universal nodes already exist in the current catalogue. Furthermore it is possible that we will need to introduce a set of domain specific universal nodes.

3. ILLUSTRATION OF THE IDEA

Let us look at the input source code [Gustafsson et al. 2009]:

```
if (x > y) z = 42;
```

Figure 1 represents corresponding eCST for this segment of source code.

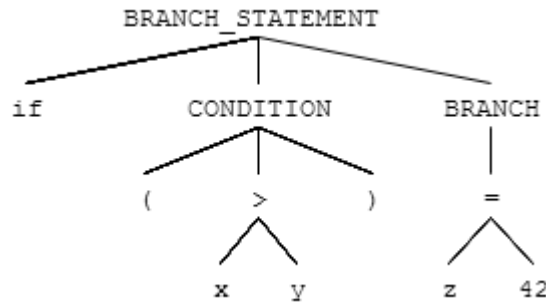


Fig. 1. eCST for if statement in C code

This segment can be translated into the ALF code below:

```

{switch    {s_le 32      {load 32 {addr 32 {fref 32 x} {dec_unsigned 32 0}}}
              {load 32 {addr 32 {fref 32 y} {dec_unsigned 32 0}}}}
  {target   {dec_unsigned 1 1}
            {label 32 {lref 32 exit} {dec_unsigned 32 0}}}}
{store     {addr 32 {fref 32 z} {dec_unsigned 32 0}}
  with {dec_signed 32 42}}
  {label 32 {lref 32 exit} {dec_unsigned 32 0}}

```

Generated segment of the ALF source code is much longer than the corresponding C code. Therefore corresponding eCST is also much larger. Figure 2 represents corresponding eCST for generated segment of ALF source code. Here we provide only the part of generated tree in order to demonstrate equivalency of trees for two languages.

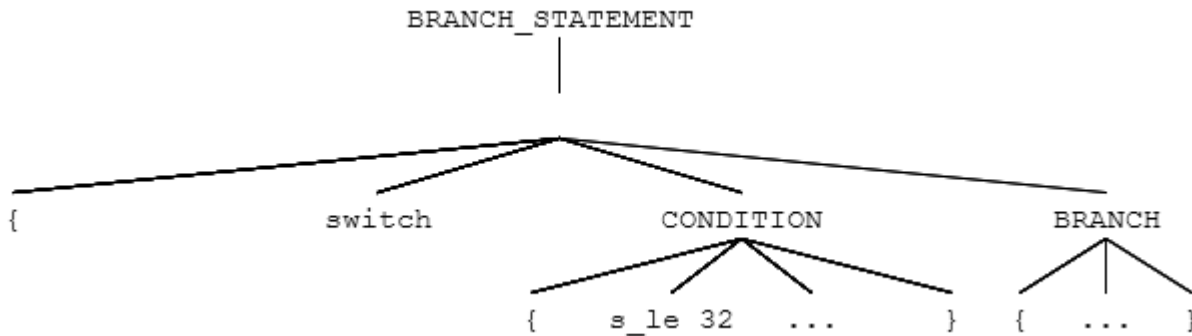


Fig. 2. Part of eCST for switch statement in ALF code

3.1 Exporting information

In this section we will demonstrate idea for extracting XML timing information. For these purposes we will take basic example of branch statement. For all other needed information we can follow similar idea to export needed data.

Based on [Parsa and Mehdi 2014] to store timing information for switch statement we need best (BTime), worst (WTime), and total (TotalTime) execution time attributes, and for each branch we need (execution) Time:

```

<SwitchBlockK BTime=n WTime=n TotalTime=n>
  <CaseBlockK Time=n>
  ...
</CaseBlockK>
...
</SwitchBlockK>

```

For If statement we need the same attributes.

```

<IfBlockK BTime=n WTime=n TotalTime=n ... >
  <ThenBlockK Time=n ... >
  ...
</ThenBlockK>
  <ElseBlock Time=n >
  ...
</ElseBlockK>
</IfBlockK>

```

In eCST representation this would be encompassed by the unique BRANCH_STATEMENT node

```

< BRANCH_STATEMENT" BTime=n WTime=n TotalTime=n>
  < BRANCH" Time=n>
  ...
  < BRANCH" Time=n>
  ...

```

Similarly we can extract all needed information from the source code to our XML representation (loops, conditions, etc.)

This XML representation will enable the flow of timing information between the tools, but also visualization of generated information, which means easier manipulation with timing facts.

4. RELATED WORK

Timing analysis and WCET calculation is a very actual research topic and many tools are currently under development. Our attention is on tools calculating WCET on the level of a source code.

aiT³ [Lokuciejewski and Marwedel 2011] is a tool for static WCET analysis and is used to compute a safe upper bound of WCET. It accepts binary executable as its input, from which the control-flow graph is reconstructed. This graph is a code representation on which several static analyses are implemented to compute the execution time of each instruction. A global path analysis is used to compute overall WCET bound of tasks.

Bound-T⁴ is a WCET static analyzer with similar characteristics. It takes machine code as input and (based on control flow paths) generates WCET bounds and (optionally) stack-usage bounds.

FORTAS (the FORMal Timing Analysis Suite)⁵ combines execution time measurements with static program analysis techniques. It estimates WCET of software tasks running on embedded real-time systems based on a hybrid approach following the general principles of measurement-based timing analysis.

SWEET (Swedish WCET Analysis Tool)⁶ is a WCET analysis tool consisting of a flow analysis, a low-level analysis, and a WCET estimation. SWEET analyzes the intermediate format ALF [Gustafsson et al. 2009]. Given a code format, SWEET can perform a WCET analysis for it if there is a translator into ALF. Therefore, SWEET currently supports C/C++, IAR, and binaries, as previously mentioned.

By integration WCET static analyzer in SSQSA framework we can expect to cover wider specter of possible inputs which is the main goal of this research.

5. CONCLUSION AND FURTHER WORK

In this paper we propose a possible approach to enable support for WCET calculation into SSQSA framework. An idea based on introducing support for domain specific language is described in order to implement and test calculation algorithm. After this task is completed, exploration of the possibilities for extending the analysis to other supported languages is needed. The aim is to enable uniform application of the same algorithm implementation to all supported languages. This task should have two phases. First phase is to support language independent flow analysis. This will be the straightforward activity as generation of all language independent code representation is already enabled. Second phase is the platform dependent generation of WCET values. This task will require deeper research on extracting specific information to eCST. It may require introducing of the domain specific universal nodes. By these nodes we would annotate domain specific information in eCST. In validation stage, gained results are to be compared with results generated by existing language specific WCET calculation tools. First level validation will be by comparing gained results with the results generated by the SWEET tool.

REFERENCES

- J. Gustafsson, A. Ermedahl, B. Lisper, C. Sandberg, L. Källberg. 2009. ALF—a language for WCET flow analysis. In *Proc. 9th International Workshop on Worst-Case Execution Time Analysis (WCET'2009)*, Dublin, Ireland, pp. 1-11
- J. Kolek, G. Rakić, M. Savić. 2013. Two-dimensional Extensibility of SSQSA Framework, In *Proceedings of the 2nd Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications*, Novi Sad, Serbia, September 15-17, 2013., pp. 35-43.
- P. Lokuciejewski, P. Marwedel. 2009. Combining Worst-Case Timing Models, Loop Unrolling, and Static Loop Analysis for WCET Minimization. In *Proceedings of the 2009 21st Euromicro Conference on Real-Time Systems (ECRTS '09)*. IEEE Computer Society, Washington, DC, USA, pp. 35-44
- P. Lokuciejewski, P., P. Marwedel, P. 2011. Worst-case execution time aware compilation techniques for real-time systems. Springer.
- S. Parsa, S. Mehdi. 2014. A XML-Based Representation of Timing Information for WCET Analysis, *Journal of mathematics and computer science*, Vol 8, Issue3, 2014, pp. 205-214.
- G. Rakić, Z. Budimac. 2013. Language independent framework for static code analysis, In *Proceedings of the 6th Balkan Conference in Informatics (BCI '13)*, Thessaloniki, Greece, September 19-21, 2013, ACM, New York, NY, USA, 236-243.
- G. Rakić, Z. Budimac. 2011., Introducing Enriched Concrete Syntax Trees, In *Proc. of the 14th International Multiconference on Information Society (IS), Collaboration, Software And Services In Information Society (CSS)*, October 10-14, 2011, Ljubljana, Slovenia, Volume A, pp. 211-214,

³ <http://www.absint.com/ait/>

⁴ <http://www.bound-t.com/>

⁵ <http://www.fortastic.net/>

⁶ <http://www.mrtc.mdh.se/projects/wcet/home.html>

- M. Savić, G. Rakić, Z. Budimac, M. Ivanović (2014), A language-independent approach to the extraction of dependencies between source code entities, *Information and Software Technology* (2014), doi: <http://dx.doi.org/10.1016/j.infsof.2014.04.011>
- R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, Per Stenstrom. 2008. The worst-case execution-time problem—overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.* 7, 3, Article 36 (May 2008), 53 pages.
- .

Graph Clustering Evaluation Metrics as Software Metrics

MILOŠ SAVIĆ and MIRJANA IVANOVIĆ, University of Novi Sad

Graph clustering evaluation (GCE) metrics quantify the quality of clusters obtained by graph clustering (community detection) algorithms. In this paper we argue that GCE metrics can be applied on graph representations of software systems in order to evaluate the degree of cohesiveness of software entities. In contrast to widely known cohesion measures used in software engineering, GCE metrics do not ignore external dependencies among software entities, but contrast them to internal dependencies to quantify cohesion. Using the theoretical framework of cohesion measurement in software engineering introduced by Briand et al. we investigate the properties of GCE metrics. Our analysis shows that GCE metrics are theoretically sound with respect to the monotonicity and merge property, but also reveals that they possess certain limitations whose importance is discussed in the paper. Finally, we propose a set of research questions for further empirical studies on this topic.

Categories and Subject Descriptors: D.2.8 [Software engineering]: Metrics—*Product metrics*

General Terms: Measurement, Theory

Additional Key Words and Phrases: cohesion, clustering, metrics

1. INTRODUCTION

Graph clustering is one of the most important method in complex network analysis [Boccaletti et al. 2006]. Identification of clusters (also known as communities or modules) in a network helps us to reduce the complexity of the network in order to be able to understand its underlying structure. Other applications of graph clustering techniques include parallel processing of graph based data, route planning, image segmentation and VLSI physical design [Buluç et al. 2013], to mention a few. Intuitively speaking, a cluster in a network is a part of the network where connections among members of the cluster are much denser than with the rest of the network [Fortunato 2010]. There is a variety of graph clustering (community detection) algorithms [Schaeffer 2007]. Naturally, they are accompanied by graph clustering evaluation (GCE) metrics that quantify the quality of partitions produced by them [Leskovec et al. 2010].

“Low coupling, high cohesion” is one of the basic design principles in software engineering [Yourdon and Constantine 1979]. This principle states that the coupling between modules of a software system has to be minimal as possible keeping at the same time strong relations between elements of each module. The main idea of this work is that highly cohesive modules that are loosely coupled to other modules can be viewed as clusters in a graph that encompasses software entities at lower levels of

This work was supported by the Serbian Ministry of Education, Science and Technological Development through project *Intelligent Techniques and Their Integration into Wide-Spectrum Decision Support*, no. OI174023.

Author’s address: M. Savić, M. Ivanović, Department of Mathematics and Informatics, Faculty of Sciences, University of Novi Sad, Trg Dositeja Obradovića 4, 21000 Novi Sad, Serbia, email: {svc, mira}@dmi.uns.ac.rs.

Copyright © by the paper’s authors. Copying permitted only for private and academic purposes.

In: Z. Budimac, T. Galinac Grbac (eds.): Proceedings of the 3rd Workshop on Software Quality, Analysis, Monitoring, Improvement, and Applications (SQAMIA), Lovran, Croatia, 19.-22.9.2014, published at <http://ceur-ws.org>.

abstraction. Therefore, the aim of this paper is to investigate existing GCE metrics as metrics reflecting cohesion of software modules.

The rest of the paper is structured as follows. Related work is presented in Section 2. Graph clustering evaluation metrics explored in this work as software metrics are defined in the next section of the paper. Section 4 explains how GCE metrics can be applied to graph representation of software systems. Theoretical analysis of GCE metrics as software metrics is given in Section 5. The last section concludes the paper and presents research questions for our future work.

2. RELATED WORK

Cohesion of a software module reflects how strongly related are the elements of the module. Perhaps the widest known cohesion metric in software engineering is LCOM (Lack of cohesion in methods) introduced by Chidamber and Kemerer [1994] in their object-oriented metrics suite. As the name of the metric suggests, LCOM is an inverse cohesion metric: a low value of LCOM indicates high cohesion of a class and vice versa. LCOM is based on a specific coupling between methods: two methods in a class are considered as data coupled if they use at least one common class attribute. Then LCOM is the number of non-coupled methods (P) reduced by the number of coupled methods (Q) if $P > Q$, or zero otherwise. From the definition of the LCOM it can be seen that this metric does not take external dependencies into account nor method invocations (another form of method coupling).

The approach of Hitz and Montazeri [1995] to measure cohesion of software entities followed the research of Chidamber and Kemerer. For a class we can construct graph G whose nodes are methods defined in the class, and two methods are connected by an undirected link if they are data coupled. LCOM of Hitz and Montazeri is the number of connected components in G . The same authors also proposed another variant of the same metric where G includes method calls relations. Finally, they introduced a metric called *connectivity* which quantifies how much G is far from being completely connected.

Bieman and Kang [1995] introduced two cohesion metrics called tight class cohesion (TCC) and loose class cohesion (LCC). The basic element in their metrics is again a graph representing a class that encompasses methods of the class. TCC/LCC is the density (the actual number of links divided by the maximal number of links) of a TCC/LCC graph. Two methods are connected in TCC graph if they both access the same variable or there is a direct call between them. LCC graph is an extension of TCC graph that includes indirect method calls.

Lee et al. [1995] introduced a class cohesion metric based on information flow. The basic idea is that the strength of call coupling between invoking and invoked method is determined by the number of parameters of invoked method: the more information passed through formal parameters, the stronger call coupling between methods. Then, the cohesion of a method is defined as the number of calls to other methods multiplied by the number of formal parameters. Finally, the cohesion of a class is the sum of cohesion of its methods.

From the review of widely used software engineering cohesion metrics it can be concluded that the cohesiveness of a software entity is estimated in isolation. In other words, those metrics rely only on internal dependencies, while external dependencies, dependencies reflecting coupling between software modules, are not taken into account. However, external dependencies can be also important when estimating cohesiveness of software modules. Firstly, a module that has much more external than internal dependencies hardly can be considered as strongly cohesive regardless of the density or the connectedness of its internal parts. Secondly, having two modules that have the same degree of internal density the one with the smaller number of external dependencies can be considered as more cohesive compared to the other.

3. GRAPH CLUSTERING EVALUATION METRICS

Let $G = (V, E)$ be a directed graph where V is the set of nodes and E set of links. Let C denote a cluster in V ($C \subseteq V$), and let c be a node from C . An intra-cluster link emanating from c connects c to another node from C , while an inter-cluster link emanating from c connects c to a node that does not belong to C . Intra-cluster out-degree of node c is the number of intra-cluster links emanating from c , while inter-cluster out-degree of node c is the number of inter-cluster links emanating from c .

The most common formulation of the graph partitioning problem asks for a division of the set of nodes into balanced, disjoint subsets of nodes such that the edge cut (links connecting nodes from different clusters) is minimized. Therefore, the basic graph clustering evaluation (GCE) metrics are based on the size of the edge cut. Let E_C denote the size of the cut (the number of inter-cluster links) for cluster C ,

$$E_C = |\{(x, y) : x \in C, y \notin C\}| = \sum_{x \in C} \text{inter-cluster out-degree}(x),$$

I_C the number of intra-cluster links for C ,

$$I_C = |\{(x, y) : x \in C, y \in C\}| = \sum_{x \in C} \text{intra-cluster out-degree}(x),$$

N_C the number of nodes in C , and N the number of nodes in the graph. Then cut based GCE metrics, conductance, expansion and cut-ratio, are defined as follows [Leskovec et al. 2010]:

- (1) **Conductance** of cluster C is the size of the cut normalized by the total number of links incident to nodes contained in C ,

$$\text{Conductance}(C) = \frac{E_C}{E_C + I_C}.$$

- (2) **Expansion** of cluster C is the size of the cut divided by the total number of nodes in C ,

$$\text{Expansion}(C) = \frac{E_C}{N_C}.$$

- (3) **Cut-ratio** of cluster C is the size of the cut divided by the size of maximal cut,

$$\text{Cut-ratio}(C) = \frac{E_C}{N_C(N - N_C)}.$$

Probably the oldest definition of graph cluster originate from circuit theory which is furtherly adopted in social network analysis. Namely, Luccio and Sami [1969] introduced the notion of LS-set that is also known as Radicchi strong community in social network analysis [Radicchi et al. 2004]. For directed graphs, an LS-set is a subgraph such that the intra-cluster out-degree of each node in the set is higher than its inter-cluster out-degree. The nodes having zero out-degree are not taken into account when determining whether the cluster is Radicchi strong. If the number of intra-cluster links is higher than the number of inter-cluster links then the subgraph is considered as Radicchi weak cluster. Each Radicchi strong cluster is at the same time Radicchi weak cluster, while the converse is not generally true. If a cluster is Radicchi weak or strong then its conductance is smaller than 0.5. The difference between the number of intra- and inter-cluster links inspired ODF (out-degree fraction) family of cluster quality measures [Leskovec et al. 2010]:

- (1) **Maximum-ODF** of cluster C is the maximum fraction of inter-cluster links of a node observed in the cluster,

$$\text{Maximum-ODF}(C) = \max_{c \in C} \frac{|\{(c, d) : d \notin C\}|}{D_{out}(c)},$$

where $D_{out}(c)$ stands for out-degree of node c .

- (2) **Average-ODF** of cluster C is the average fraction of inter-cluster links of nodes from C ,

$$\text{Average-ODF}(C) = \frac{1}{N_C} \sum_{c \in C} \frac{|\{(c, d) : d \notin C\}|}{D_{out}(c)}$$

- (3) **Flake-ODF** of cluster C is the fraction of nodes in C that have higher intra-cluster out-degree than inter-cluster out-degree,

$$\text{Flake-ODF}(C) = \frac{|\{x : x \in C, |\{(x, y) : y \notin C\}| < D_{out}(x)/2\}|}{N_C}.$$

In other words Flake-ODF measures how C is close to being Radicchi strong cluster: if Flake-ODF(C) is equal to 1 then C is Radicchi strong.

4. SOFTWARE NETWORKS AND CLUSTERING METRICS

Software networks are graph-based representations of a software system. The architecture of the whole system can be represented by one directed graph that we refer to as a General Dependency Network (GDN) [Savić et al. 2014]. The nodes of GDN represent software entities such as packages/units, classes/modules, methods/functions and class attributes/global variables, while links represent relations between them. We can distinguish between two types of links in GDN: “vertical” (CONTAINS) links that maintain the hierarchy of software entities and “horizontal” links that show dependencies between entities from the same level of abstraction. Two software entities A and B are connected by a CONTAINS link $A \rightarrow B$ if entity A defines or declares entity B . A group of entities that are contained in the same highly cohesive and loosely coupled entity naturally form a cluster of contained entities. Examples of such clusters for object-oriented software systems are: (1) classes and interfaces contained in the same package or workspace, and (2) methods and class attributes contained in the same class. When a software is written in a procedural programming language then procedures (functions) and global variables defined in a module form a cluster.

We can separate horizontal links of GDN into two categories:

- Intra-cluster link connects two entities from the same level of abstraction that are contained in the same software entity, i.e.

$$A \rightarrow B \text{ is an intra-cluster link} \Leftrightarrow (\exists O) \text{CONTAINS}(O \rightarrow A) \wedge \text{CONTAINS}(O \rightarrow B).$$

- Inter-cluster link connects two entities from the same level of abstraction that are contained in two different software entities, i.e.

$$A \rightarrow B \text{ is an inter-cluster link} \Leftrightarrow (\exists O_1, O_2) O_1 \neq O_2 \wedge \text{CONTAINS}(O_1 \rightarrow A) \wedge \text{CONTAINS}(O_2 \rightarrow B).$$

The separation of links into intra- and inter-cluster links enables us to apply graph clustering evaluation (GCE) metrics to:

- (1) Class collaboration networks in order to evaluate cohesiveness of packages. Class collaboration network is a subgraph of GDN that shows dependencies between classes and interfaces.
- (2) Extended static call graphs in order to evaluate cohesiveness of classes in OO systems or modules in procedural software systems. Functions (methods) and global variables (class attributes) constitute the set of nodes in an extended static call graph, while links denote call relationships between functions and uses (access) relationships between functions and global variables.

It can be easily seen from the definition of GCE metrics that only the Flake-ODF metric measures cohesion, while other metrics introduced in the previous section are inverse cohesion measures. In

contrast to cohesion metrics widely used in software engineering (see Section 2), GCE metrics do not ignore references to external entities. On the contrary, they use the number of dependencies to external references to determine to what extent the entity is isolated from the rest of the system. In other words, GCE metric are based on the following principle: an entity can be considered as highly cohesive if its elements are better connected themselves than with the entities defined outside the entity.

Figure 1 shows a class collaboration network that represent a simple software system that consists of two packages P and Q where both packages contain three classes. It can be observed that class F has higher inter-cluster out-degree than intra-cluster out-degree: this class references one class from its package and two classes from package P . Therefore, package Q is not Radicchi strong cluster. This package is neither Radicchi weak cluster since the number of intra-cluster links is not higher than the number of inter-cluster links. It can be also observed that the system presented in Figure 1 can be refactored in order improve the overall degree of cohesion: if we move class F from package Q to package P then both packages will be Radicchi strong.

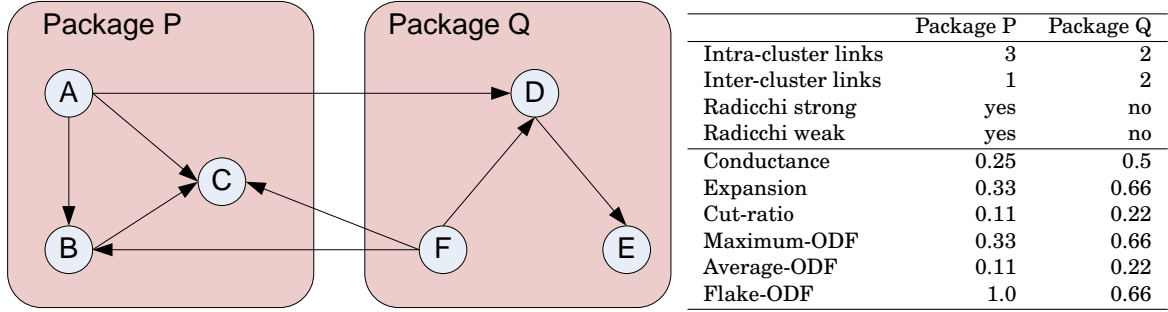


Fig. 1. Class collaboration network of a simple software system and appropriate cluster quality measures.

5. THEORETICAL ANALYSIS

Briand et al. [1996; 1998] defined several properties that a software metric should satisfy in order to be theoretically sound (lack of) cohesion metric. Those properties are:

- (1) **Nonnegativity.** A cohesion (lack of cohesion) metric cannot take a negative value.
- (2) **Normalization.** The metric belongs to an interval $[0, M]$, where M is the fixed maximal value.
- (3) **Null value.** The cohesion of a software entity is null if R_c is empty, where R_c denotes the set of relationships within the software entity. This means that if there are no intra-cluster links the cohesion of the entity should be zero. On the other side, a metric measuring the lack of cohesion should be zero if R_c is maximal. R_c is maximal if all possible relationships within the entity are present.
- (4) **Maximum value.** If R_c is maximal then a metric of cohesion takes the maximal value. If $R_c = \emptyset$ then a metric measuring the lack of cohesion takes the maximal value.
- (5) **Monotonicity.** Let e be a software entity. Let e' be the software entity such that $R_e \subseteq R_{e'}$, i.e. we added some relationships (intra-cluster links) in e to obtain e' . Then the following inequalities must hold

$$C(e) \leq C(e'), \quad (1)$$

$$L(e) \geq L(e'), \quad (2)$$

where C and L denote a cohesion and lack of cohesion metric, respectively. In other words, the property states that addition new intra-cluster links must not decrease/increase the value of the cohesion/lack of cohesion metric.

- (6) **Merge property.** Let e_1 and e_2 be two unrelated (unconnected) software entities. This means that e_1 does not reference e_2 and vice versa, i.e. there are no relationships (inter-cluster links) between e_1 and e_2 . Let e be the software entity which is the union of e_1 and e_2 . Then the following inequalities must hold

$$C(e) \leq \max\{C(e_1), C(e_2)\}, \quad (3)$$

$$L(e) \geq \min\{L(e_1), L(e_2)\}. \quad (4)$$

Namely, this property says that merging two unrelated entities must not increase/decrease the value of the cohesion/lack of cohesion metric.

As a first step in our theoretical analysis of graph clustering evaluation metrics, we state and prove the following lemma that will be frequently used in this section.

LEMMA 1. *Let P and Q be two nonnegative numerical properties of a module. If P and Q are additive under the merge operation then a (lack of) cohesion metric defined as $C = P/Q$ satisfies the merge property.*

PROOF. Let m_1 and m_2 be two modules. Without loss of generality we can assume that $C(m_1) \leq C(m_2)$. Due to the nonnegativity of P and Q the following inequality holds

$$P(m_1)Q(m_2) \leq P(m_2)Q(m_1). \quad (5)$$

Let m denote the module obtained by merging m_1 and m_2 . Due to the additivity of P and Q we have that

$$C(m) = \frac{P(m_1) + P(m_2)}{Q(m_1) + Q(m_2)}.$$

C is a cohesion metric. Let us suppose that the merge property is not satisfied, i.e.

$$C(m) > \max\{C(m_1), C(m_2)\} = C(m_2).$$

By elementary algebraic transformation we obtain that

$$P(m_1)Q(m_2) > P(m_2)Q(m_1) \quad (6)$$

which is in contradiction with inequality 5.

C is a lack of cohesion metric. Again we give a proof by contradiction. If

$$C(m) < \min\{C(m_1), C(m_2)\} = C(m_1)$$

then by elementary algebraic transformation we again obtain inequality 6. \square

From the definition of GCE metrics (see Section 3) it can be easily seen that all of them are non-negative. The maximal value of conductance is equal to 1 when $R_c = \emptyset$ and consequently this measure satisfies both the normalization property and the maximum value property. When R_c is maximal conductance is not necessarily equal to zero. Namely, conductance is equal to zero if and only if a module does not depend on other modules. Adding intra-cluster relationship increases only the denominator of conductance and consequently conductance satisfies the monotonicity property. The merge property of conductance is the consequence of Lemma 1 when P is the number of inter-cluster links and Q the sum of the number of inter- and intra-cluster links. Namely, the number of intra-cluster links is an

additive property under the merge operation. Secondly, if two modules are unrelated then they have disjoint sets of inter-cluster links. This means that the number of inter-cluster links is also an additive property for unrelated modules.

In contrast to conductance, expansion does not satisfy the normalization property. If we modify expansion to be a value in the interval $[0,1]$ then we actually obtain the cut-ratio metric. Expansion also does not satisfy the null value property and the maximum value property: both the numerator and denominator in the definition of expansion are independent on the number of intra-cluster links. The expansion of a module remains the same under the addition of intra-cluster links. Therefore, this metric also satisfies the monotonicity property. As already mentioned, the number of inter-cluster links is an additive property of disjoint modules. The number of nodes in a module is also an additive property under the merge operation. Therefore, by Lemma 1 expansion satisfies the merge property.

Cut-ratio satisfies the normalization property: the maximal value of cut-ratio is equal to 1 which is obtained when each entity from the module references all entities defined outside the module. Both the numerator and the denominator of cut-ratio are independent of the number of intra-cluster links and similarly as expansion this measure does not satisfy the null and the maximum value property. If we add a new intra-cluster link the cut-ratio does not change and consequently this metric satisfies monotonicity property. The cut-ratio metric satisfies the merge property which shows the following lemma.

LEMMA 2. *Cut-ratio satisfies the merge property.*

PROOF. Let C_x denote the number of inter-cluster links emanating from nodes contained in module x , N_x the number of nodes in module x , and N the number of nodes in the whole network. Let p and q be two disconnected modules such that the cut-ratio of p is smaller than the cut-ratio of q , i.e.

$$\frac{C_p}{N_p(N - N_p)} \leq \frac{C_q}{N_q(N - N_q)} \quad \Leftrightarrow \quad C_p N_q (N - N_q) \leq C_q N_p (N - N_p). \quad (7)$$

Let r denote the union of p and q . Let us suppose that the merge property is not satisfied, i.e.

$$\frac{C_p + C_q}{(N_p + N_q)(N - N_p - N_q)} < \frac{C_p}{N_p(N - N_p)} \quad (8)$$

If we multiply both sides of inequality 8 by $(N_p + N_q)(N - N_p - N_q)N_p(N - N_p) > 0$, then we obtain

$$(C_p + C_q)N_p(N - N_p) < C_p(N_p + N_q)(N - N_p - N_q) \quad (9)$$

$$C_q N_p (N - N_p) < C_p N_q (N - N_q) - 2C_p N_p N_q \quad (10)$$

$$\leq C_p N_q (N - N_q) \quad (11)$$

which is in contradiction with inequality 7. \square

From the definition of ODF measures it can be easily seen that they take values in the range $[0, 1]$, which means that they satisfy both nonnegativity and normalization property. When $R_c = \emptyset$ then Maximum-ODF and Average-ODF are equal to 1, while Flake-ODF is equal to 0, which means that Maximum- and Average-ODF satisfy the maximum value property, while Flake-ODF satisfies the null value property (recall that Flake-ODF measures cohesion, while Maximum- and Average-ODF are lack of cohesion metrics). The numerator of Maximum- and Average-ODF is independent of the number of intra-cluster links. Consequently, those metrics do not satisfy the null value property and satisfy the monotonicity property (addition of intra-cluster links does not change Maximum- and Average-ODF). The merge property is trivially satisfied for Maximum-ODF.

LEMMA 3. *Average-ODF satisfies the merge property.*

PROOF. Let D'_a denote the number of inter-cluster links emanating from node a , D_a out-degree of node a ($D'_a \leq D_a$), and N_x the number of nodes in module x . Let p and q be two disconnected modules such that the Average-ODF of p is smaller than the Average-ODF of q , i.e.

$$\frac{1}{N_p} \sum_{u \in p} \frac{D'_u}{D_u} \leq \frac{1}{N_q} \sum_{u \in q} \frac{D'_u}{D_u} \Leftrightarrow N_q \alpha \leq N_p \beta, \quad \text{where } \alpha = \sum_{u \in p} \frac{D'_u}{D_u}, \beta = \sum_{u \in q} \frac{D'_u}{D_u} \quad (12)$$

Let r denote the union of p and q . The Average-ODF of r is equal to

$$\text{Average-ODF}(r) = \frac{1}{N_p + N_q} \sum_{u \in r} \frac{D'_u}{D_u} = \frac{1}{N_p + N_q} \left(\sum_{u \in p} \frac{D'_u}{D_u} + \sum_{u \in q} \frac{D'_u}{D_u} \right) = \frac{\alpha + \beta}{N_p + N_q} \quad (13)$$

Let us suppose that Average-ODF does not satisfy the merge property, i.e. $(\alpha + \beta)/(N_p + N_q) < \alpha/N_p$. Then we obtain that $\beta N_p < \alpha N_q$ which is in contradiction with inequality 12. \square

The addition of new intra-cluster links can only increase the number of entities defined in a module whose intra-cluster out-degree is greater than inter-cluster out-degree. Therefore, Flake-ODF satisfies the monotonicity property. The number of entities in the module whose intra-cluster out-degree is greater than inter-cluster out-degree is additive property under the merge operation. Therefore, Flake-ODF also satisfies merge property by Lemma 1.

Table I. Properties of graph clustering metrics as (lack of) cohesion software metrics.

Metric	Nonnegativity	Normalization	Null value	Maximum value	Monotonicity	Merge
Conductance	yes	yes	no	yes	yes	yes
Expansion	yes	no	no	no	yes	yes
Cut-ratio	yes	yes	no	no	yes	yes
Maximum-ODF	yes	yes	no	yes	yes	yes
Average-ODF	yes	yes	no	yes	yes	yes
Flake-ODF	yes	yes	yes	no	yes	yes

The properties of graph clustering metrics as (lack of) cohesion software metrics are summarized in Table I. This table indicates the limitations of GCE metrics as software metrics. As observed by Briand et al. [1998], only a few widely known software cohesion metric fulfill all of the cohesion properties. In other words, a measure which does not satisfy all of the properties can be considered as poorly defined. Secondly, we can see that GCE metrics reflecting cohesion does not satisfy the null value property, while GCE metrics reflecting lack of cohesion does not satisfy the maximum value property. However, we believe that this is not the disadvantage of GCE metrics. Firstly, it is very unlikely to observe full connected software modules in practice (each class from a package reference each other; each method from a class calls each other and access to each class attribute). Secondly, in such cases GCE metrics favour loosely coupled software modules emphasizing another quality principle of good software design, i.e. the principle of low coupling.

6. CONCLUSION AND FUTURE WORK

In this paper we introduced the idea of applying graph clustering evaluation (GCE) metrics to graphs representing software systems in order to evaluate cohesiveness of software entities. In contrast to standard cohesion metrics, GCE metrics do not ignore external references. They are based on the idea that reducing coupling between an entity and the rest of the system increases cohesion of the elements contained in the entity. Using the theoretical framework introduced by Briand et al. we investigated the properties of graph clustering evaluation metrics. This analysis showed that GCE metrics are theoretically sound with respect to the most important properties of software cohesion

metrics (monotonicity and merge property), but also showed that they possess certain limitations we should be aware of when using GCE metrics as software metrics. Our future work will extend the present work with an empirical investigation of the following research questions:

- (1) Do GCE metrics correlate to standard software cohesion metrics (LCOMs, TCC, LCC, etc.) and to what extent?
- (2) Each software entity can be described by a numerical vector containing metrics of internal complexity (such as LOC, Halstead measures, cyclomatic complexity, etc.) and metric of design complexity (metrics quantifying importance of the entity such as betweenness centrality and page rank, its coupling to other entities such as degree centrality/CBO, inheritance for classes such as NOC and DIT, and invocation for methods/functions). Each of these vectors can be, according to the degree of cohesion, classified as Radicchi strong (strongly cohesive), Radicchi weak (weakly cohesive) or poorly cohesive (entity that is neither Radicchi strong nor Radicchi weak). Therefore, our second research question will be: are there any differences in internal and design complexity between strongly, weakly and poorly cohesive software entities?
- (3) Is it possible to automatically remodularize software system using simple refactorings such as move class/method in order to improve the degree of cohesion of the overall system (to increase the number of Raddichi strong clusters, to minimize the average conductance, etc.).

REFERENCES

- James M. Bieman and Byung-Kyoo Kang. 1995. Cohesion and Reuse in an Object-oriented System. In *Proceedings of the 1995 Symposium on Software Reusability (SSR '95)*. ACM, New York, NY, USA, 259–262.
- S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, and D-U. Hwang. 2006. Complex Networks : Structure and Dynamics. *Physics Reports* 424, 4-5 (2006), 175–308.
- Lionel C. Briand, John W. Daly, and Jürgen Wüst. 1998. A Unified Framework for Cohesion Measurement in Object-Oriented Systems. *Empirical Software Engineering* 3, 1 (1998), 65–117.
- Lionel C. Briand, Sandro Morasca, and Victor R. Basili. 1996. Property-Based Software Engineering Measurement. *IEEE Transactions on Software Engineering* 22, 1 (1996), 68–86.
- Aydin Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. 2013. Recent advances in graph partitioning. *CoRR* abs/1311.3144 (2013).
- S. R. Chidamber and C. F. Kemerer. 1994. A metrics suite for object oriented design. *IEEE Transactions in Software Engineering* 20, 6 (1994), 476–493.
- Santo Fortunato. 2010. Community detection in graphs. *Physics Reports* 486, 3-5 (2010), 75 – 174.
- Martin Hitz and Behzad Montazeri. 1995. Measuring Coupling and Cohesion in Object-Oriented Systems. In *Proc. International Symposium on Applied Corporate Computing*. 25–27.
- Y. S. Lee, B. S. Liang, S. F. Wu, and F. J. Wang. 1995. Measuring the coupling and cohesion of an object-oriented program based on information flow. In *Proceedings of International Conference on Software Quality*.
- Jure Leskovec, Kevin J. Lang, and Michael Mahoney. 2010. Empirical Comparison of Algorithms for Network Community Detection. In *Proceedings of the 19th International Conference on World Wide Web (WWW '10)*. ACM, New York, NY, USA, 631–640.
- F. Luccio and M. Sami. 1969. On the decomposition of networks in minimally interconnected subnetworks. *IEEE Transactions on Circuit Theory* 16, 2 (1969), 184–188.
- F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi. 2004. Defining and identifying communities in networks. *Proceedings of the National Academy of Sciences* 101, 9 (2004), 2658–2663.
- Milos Savić, Gordana Rakić, Zoran Budimac, and Mirjana Ivanović. 2014. A language-independent approach to the extraction of dependencies between source code entities. *Information and Software Technology*, 56, 10 (2014), 1268–1288.
- Satu Elisa Schaeffer. 2007. Graph Clustering. *Compututer Science Review* 1, 1 (2007), 27–64.
- Edward Yourdon and Larry L. Constantine. 1979. *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design* (1st ed.). Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Approaches for Test Case Generation from UML Diagrams

TINA SCHWEIGHOFER and MARJAN HERIČKO, University of Maribor

Model based testing (MBT) is an important approach with many advantages that can reduce the cost and increase the effectiveness and quality of a testing procedure. In MBT, test cases can be derived from different models, also from the popularly used UML diagrams. Different UML diagrams include various important pieces of information that can be successfully used in a testing procedure. A lot of papers present approaches for test case generation from different UML diagrams and researchers are trying to find the most optimal one. In this paper, we present the first results of a systematic literature review in the area of test case generation from UML diagrams. Based on research questions, we explored which UML diagrams are most commonly used for test case generation, what approaches are presented in the literature, their pros and cons and connections with different testing levels. We also tried to find approaches that are tailored to test mobile applications. First results show that UML state machine, activity, sequence diagram and of course, a combination of more UML diagrams, are most commonly used for test case generation. Different approaches are used for generation, like graphs, trees, tables, labelled transition systems (LTS), genetics algorithms (GA), finite state machines (FSM) and others. The found approaches have many advantages, but also some disadvantages such as a lack of automatization, problems with complex diagrams and others. A detailed analysis is presented in this article.

General Terms: Test case generation from UML

Additional Key Words and Phrases: test case generation, UML, approaches, MBT, state machine diagram, activity diagram, sequence diagram, SLR

1. INTRODUCTION

Testing is an important part of the software development process in which we want to check if a product satisfies the given requirements. It is a non-trivial process with many important parts. As products become increasingly more complex, the process has become very extensive and time consuming. Consequently, a logical conclusion would be the automatization of the process. Not only for the execution of test cases, but also the preparation process, which includes test case design and generation. The topic of test case generation is becoming more and more popular and because test case design and execution are time and resource consuming, it is understandable that automatic test case generation constitutes an important topic [Samuel et al. 2008].

Test cases can be generated from code, graphs, formal specifications and different models. Testing from models, also known as model based testing (MBT), is currently a popular research topic. MBT is a testing methodology that usually facilitate the automation of a test case generation using either models or properties, as the basis for deriving complete test suites [Francisco and Castro 2012].

One of the most popular models used for test case generation are UML diagrams. The Unified Modeling Language (UML) is used for modeling and presents different views of the system. Some of the diagrams are very popular and often used for modeling; the same can be observed in the process of test case generation. UML models constitute an important source of information for test case design. Therefore, UML based automatic test case generation is an important but also theoretically challenging topic that has been getting a lot of attention from researchers.

The generation of test cases from UML diagrams is also a research topic of our paper. We want to explore which UML diagrams are most commonly used for the test case generation process, how widespread the approaches are and which techniques are used. We used the research method systematic literature review (SLR) and followed good practices and recommendations. Detailed research questions were formed and a detailed analysis of the approaches were made with an emphasis on their pros, cons,

Author's address: T. Schweighofer, Faculty of Electrical Engineering and Computer Science, University of Maribor, Smetanova 17, 2000 Maribor, Slovenia; email: tina.schweighofer@um.si; M. Heričko, Faculty of Electrical Engineering and Computer Science, University of Maribor, Smetanova 17, 2000 Maribor, Slovenia; email: marjan.hericko@um.si.

Copyright © by the paper's authors. Copying permitted only for private and academic purposes.

In: Z. Budimac, T. Galinac Grbac (eds.): Proceedings of the 3rd Workshop on Software Quality, Analysis, Monitoring, Improvement, and Applications (SQAMIA), Lovran, Croatia, 19.-22.9.2014, published at <http://ceur-ws.org>

and opportunities for improvement. The content of the work is organized as follows: After the introduction, some theoretic background is presented; including test case generation, MBT and UML, followed by the research part of the work. The research method is described and the results are presented in an organized form. At the end, results are discussed and summarized and the paper is concluded.

2. BACKGROUND

2.1 Test Case Generation

The testing effort is divided into three parts: test case generation, test execution and test evaluation. In comparison with the other two parts, test case generation is the most challenging [Gulia and Chillar 2012; Mingsong et al. 2006]. Test cases that are created manually are usually error prone and time consuming, so the automation of test case specifications is the next logical phase [Schwarzl and Peischl 2010]. Test case generation can save time and effort and at the same time reduce the number of errors and faults [Gulia and Chillar 2012]. It also cuts down on the costs of manual testing and increases the reliability of tests [Shamsoddin-Motlagh 2012]. The reason lies in test cases that can be generated from models in parallel with the implementation of system, which can then be easily updated if the specifications change [Dalal et al. 1999; Wang et al. 2008].

2.2 Model Based Testing

Model based testing (MBT) is a promising approach for software quality control and for reducing the costs of a test process, because test cases can be generated from the software specifications at the same time as development [Cartaxo et al. 2007; Zeng et al. 2009]. MBT can be described with the following action. First, the model is built from software requirements, and the expected inputs and outputs are generated from a formal model. Tests are then run and inputs and outputs are generated. Finally, those outputs are compared to the expected outputs [Cartaxo et al. 2007].

MBT generally creates tests from an abstract model of the software, including formal specifications and semi-formal design descriptions such as UML diagrams [Kansomkeat et al. 2008]. It relies on behaviour models based on input and expected output for test case generation in its implementation [Pretschner et al. 2005]. MBT has evolved out of techniques like finite state machine (FSM), labelled transition systems (LTS), Message Sequence Charts (MSC) and Petri nets to generate test cases for systems [Shirole and Kumar 2013]. One of the most used and popular models for MBT are also UML diagrams [Swain et al. 2010].

2.3 Unified Modeling Language

In our research, we focused on the three UML diagrams that are most commonly used for test case generation: the UML state machine diagram, the UML activity diagram and the UML sequence diagram. UML state machine diagrams are part of behaviour diagrams and are well suited for describing the behaviour of a system [OMG 2011; Schwarzl and Peischl 2010]. It models dynamic behaviour and captures different states that an object can be in and its response to various events that may arise in each of its states [OMG 2011; Samuel et al. 2008]. Because there are formalized aspects of UML, a state diagram can provide a natural basis for test data generation [Shirole et al. 2011]. An activity diagram is a UML diagram that provides a view of the behaviour of a system by describing the sequence of actions in a process [Fan et al. 2009]. An activity diagram is used to depict all possible flows of execution in a use case [Nayak and Samanta 2011]. A sequence of activities in an activity diagram of a use case can be used to generate test cases. It is required to identify all possible begin-to-end paths in an activity diagram in order to cover all the activities and flow of constructs to test a use case satisfactorily [Nayak and Samanta 2011]. The sequence diagram is known as an interaction diagram that represents a scenario as a possible sequence of messages that are exchanged among the object [Khandai et al. 2011].

3. RESEARCH METHOD

Based on a set problem and desired results, we decided to use the research method systematic literature review (SLR). SLR is a means of identifying, evaluating and interpreting all available research relevant to a particular research question or topic area or phenomenon of interest. Individual studies contributing to a SLR are called primary studies and a systematic review is a form of secondary study [Kitchenham and Charters 2007].

3.1 Research Questions

Specifying research questions is the most important part of any systematic review. The review questions drive the entire systematic review methodology [Kitchenham and Charters 2007]. In our research we form 3 mayor research questions that are presented below.

RQ1: *Are there any systematic literature reviews or mapping studies on the topic of test case generation from UML diagrams?*

RQ1.1: *Are there any systematic literature reviews or mapping studies on the topic of test case generation?*

RQ2: *Is there any research regarding test case generation or model based testing from UML in the field of mobile applications?*

RQ3: *Are there any studies regarding test case generation from different UML models?*

RQ3.1: *Which types of UML diagrams are used for test case generation?*

RQ3.2: *What approaches of test case generation are used?*

RQ3.3: *What are the pros and cons of the used approaches?*

RQ3.4: *What are the connections between UML diagram type and testing approach?*

3.2 Data sources and Search Strings

To properly perform a literature review, we chose the appropriate data sources. We decided that we would use different Digital Libraries. Some of the available electronic bases were then searched for primary studies. Table I shows the Digital Libraries that were used.

A general approach to define search strings is to break down research questions into individual terms [Kitchenham and Charters 2007]. In our case, terms were then combined with the logical operator “AND” in order to link together different terms. Different search strings defined from the defined research questions are shown in Table II with links to the appropriate research question.

Table I. Data Sources for SLR

ELECTRONIC BASE	URL
IEEE Xplore	http://ieeexplore.ieee.org
ScienceDirect	http://www.sciencedirect.com
SpringerLink	http://link.springer.com
ACM Digital Library	http://dl.acm.org
Scopus	http://www.scopus.com
ProQuest	http://search.proquest.com
EBSCO DiscoveryService	http://eds.a.ebscohost.com/

Table II. Defined Search Strings

RESEARCH QUESTION	SEARCH STRING
RQ1 and RQ 1.1	"test case" AND "generation" AND "review" AND "systematic" AND "UML" "test case" AND "generation" AND "mapping" AND "systematic" AND "UML" "test case" AND "generation" "test case generation" "test case" AND "generation" AND "review" AND "systematic" "test case" AND "generation" AND "mapping" AND "systematic"
RQ2	"test case" AND "generation" AND "mobile" "model based testing" AND "mobile"
RQ3 and RQ 3.1-3.4	"test case" AND "generation" AND "UML" "model based testing" AND "UML"

3.3 Search Results and Study selection

First, a search was conducted using the search string. A different number of results were found for different search strings. When there were too many results, we limited our search on just the abstracts, title and keywords. The results were reviewed and potentially relevant primary studies were gathered. After this, studies were assessed for their actual relevance with the aim of a study selection to identify those primary studies that provide direct evidence about research questions [Kitchenham and Charters 2007]. We reviewed the selected studies and appropriate ones were selected for further reading. In order to determine whether or not the study was selected, the title and abstract were evaluated regarding inclusion and exclusion criteria. Selection criteria were set in order to reduce the likelihood of bias. They were based on research questions [Kitchenham and Charters 2007]. Exclusion criteria were that the paper was not available in selected electronic databases, the paper is not in English and paper that does not describe the test case generation process. On the other hand, inclusion criteria was that the focus is on generating test cases and that the paper deals with test case generation from the UML model. After initial studies were selected, they were reviewed in detail and the whole article was examined. In the second round, 67 studies were selected for further examination.

3.4 Data Extraction and Data Synthesis

After the study selection, we performed data extraction from the selected primary studies. Data was collected using a form that was designed and reviewed to collect all the information needed to address the review question. The data fields that we collected are presented in Table III.

Data synthesis involves collating and summarising the results of the included primary studies [Kitchenham and Charters 2007]. 67 primary studies were selected for further analysis. We decided that we would continue the analysis separately depending on the type of UML diagram used for test case generation. We decided on the following categories, named after UML diagram types: UML state diagram, UML activity diagram, UML sequence diagram and different used UML diagrams. We are aware that there are many different types of UML diagrams, but those that we selected are used in a significant number of articles. Some others, for example the collaboration or class diagram, were rarely used as only a diagram for test case generation. This is why only the previously mentioned UML diagram types were examined in detail and, of course, papers with a combination of different UML diagrams.

Table IV gives the exact number of selected primary studies divided by types of UML diagrams.

Table III. Data Extraction Form

DATA TYPE	MEANING
Title	Title of the paper.
Authors	Authors of the paper.
Year	Year the paper was published.
UML diagram type	UML diagram used for test case generation.
Testing level	Testing level covered by the approach.
Approach	Describing the proposed approach for test case generation.
Complementary techniques	Used the complementary techniques in the approach.
Practical Example	Simple example, prototype or implementation of approach.
Purpose	Purpose of the approach.
Pros	Approach pros.
Cons	Approach cons.
Future work	Future work described in the article.
Notes	Notes taken by the reviewer.

Table IV. Number of Selected Primary Studies by Types of UML diagram

	STATE MACHINE DIAGRAM	ACTIVITY DIAGRAM	SEQUENCE DIAGRAM	COMBINED DIAGRAMS
Number of studies	22	19	7	19

4. RESULTS

Much of the literature in the field of test case generation is available from different UML diagrams. The literature presents different techniques and approaches, where each has its own advantages and disadvantages. Many examples are found in the literature that describe test case generation from a single UML model and most commonly used are the diagram techniques that we choose and are presented in Table IV. We made an analysis based on data that was extracted and the results show the basis for our answers to specific research questions.

4.1 RQ1

In the first research question, we were looking for systematic studies regarding test case generation from UML diagrams. We found research [Shamsoddin-Motlagh 2012] that addressed automatic test case generation and presented approaches based on UML, graphs, formal methods, web applications and web services. They list some of the techniques but we focused mostly on UML based approaches. Some papers were also found that cover only the area of test case generation from UML diagrams. The most broad one was presented by the authors [Kaur and Vig 2012]. They present a systematic survey of the work done in the field of the automatic generation of test cases. Many techniques proposed for test case generation were found based on different UML techniques. The article [Shirole and Kumar 2013] presents a survey which aims to improve the understanding of UML behavioural based techniques. They present approaches for test case generation based on UML Sequence, State Chart and Activity diagram. We also came across an article [Aggarwal and Sabharwal 2012] that presents only approaches for test case generation from UML State Machine diagrams. Different techniques were presented and divided into groups, based on different methods usage.

4.2 RQ2

The second question was aimed at finding any available research regarding test case generation in the area of mobile applications. We found one piece of research [Chouhan et al. 2012] that directly addresses test case generation from UML activity diagram for mobile applications. The diagram was converted into a table and then into a graph by an algorithm. The test cases are then generated. In our view, the approach does not cover any special properties of mobile applications in the test case generation process. It only addresses the problem of many pages linked with each other and some predefined sequences of occurrences. Another approach is presented by [Cartaxo et al. 2007]. They present a systematic procedure

of functional test case generation by mobile phone applications. They are focused on testing features, which is an increment of functionality and they call mobile applications features. They propose an example of a feature, like Message, that has “send” and “receive” functionalities. The work is part of the research for Motorola mobile phone applications. Their approach is tailored for testing mobile applications or features, whose requirements are specified by sequence diagrams. But, the features (what they call mobile applications) are not the mobile applications that we know in a modern context.

4.3 RQ3

The next RQ is aimed at researching different approaches for test case generation from UML diagrams. We tried to find studies in this area, revealed which UML models are used with which techniques and what are pros and cons of the approaches.

We can surely confirm this, because in the process of searching for appropriate studies we found a lot of results. All of them were not appropriate, so a few of them were eliminated. In the end we chose 67 appropriate studies. The distribution of articles according to different techniques have already been presented in Table IV. An analysis of publications per year by different UML diagrams is presented in Figure 1.

We concluded that the most common diagrams used for test case generation are: the UML state machine diagram, UML activity diagram, UML sequence diagram and a combination of others diagrams like UML class, object and use case diagrams. The same conclusion was also made in the article [Kaur and Vig 2012]. They found that the most widely used ones were a combination of different UML techniques, then UML state, activity, and sequence diagram. UML class, object and use case diagrams are not precise enough for MBT according to [Shirole and Kumar 2013; Utting and Legeard 2007]. Hence, an additional description from the dynamic behaviour model was needed [Shirole and Kumar 2013; Utting and Legeard 2007]. That is why these types of diagrams are often used in test case generation examples as the complementing diagram technique.

We can conclude that regardless of the type of UML diagram, there are some joint techniques used for test case generation, like graphs, diagrams, trees and tables, labelled transition systems, genetic algorithms, OCL, XML and XMI. One widespread technique is also the finite state machine (FSM), but it can be found only in the area of the UML state machine diagram. This is not surprising, because FSM provides basic mathematical concepts for the UML state machine diagram.

Analysed approaches have a number of disadvantages. They are the same ones that plague the UML technique in general. In every analysed group, we found the same problems, like a large number of test cases and test sequences to achieve a good result, while only simple diagrams can be used, where all elements are not supported and specific issues like polymorph, concurrency and infinity loop are not addressed. Also, all the approaches are not automated, some steps have to be done manually and information from some types of diagrams are not enough. Some approaches are expensive and others are in the early stages of maturity and not tested enough. Some are presented only as a theoretical approach with some case studies and examples. We also see that when tools are presented, the test case generation process usually is not well represented.

However, there are also some advantages for the approaches. In general, all of them are trying to solve problems regarding manual testing and long and expensive test case generation procedures. They try to solve problems regarding test case coverage problems and reduce human errors. In addition, some approaches try to improve previously presented approaches. Approaches can process a large amount of data, so that a larger system can also be tested efficiently and controlled.

Despite that, we found approaches that generate test cases for almost all levels and types of testing. There are some findings about which UML diagram is most suitable for each type of testing.

If we start with the UML state machine diagram, we found approaches appropriate for unit testing, system level testing, conformance, and functional testing. According to the results, we can conclude that the majority of approaches are meant for unit level testing. This is also confirmed in different studies [Khandai et al. 2011; Kansomkeat et al. 2008], where they found out that UML state machine diagrams are the most suitable for deriving test cases for unit testing. The second diagram we analysed is UML

activity diagram. Studies revealed that the test cases generated can be used for system level testing and some of them also for functional testing. The last diagram technique is the UML sequence diagram. Test cases can be used for different testing levels and types, like unit, system testing and functional and conformance testing. But, as shown in the research [Kansomkeat et al. 2008] UML Sequence diagrams are very useful for integration level testing.

In approaches that combine different UML diagram techniques, we primarily found approaches for system level testing those that represent implemented tools. But, there are also approaches that generate test cases for unit and integration level testing.

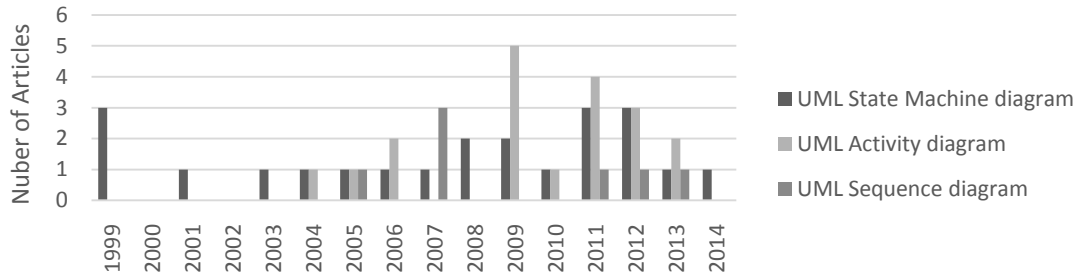


Fig 1. Publications per year divided by different UML diagrams

5. DISCUSSION

Test case generation from different UML diagrams constitutes the main research topic of the presented work. The scope of the research is broadly explored and presented in many articles and studies available from different sources. Our goal was to investigate the field and answer the proposed research questions. Using the research method SLR, we developed a research protocol. After we selected primary studies appropriate for further research, we performed an analysis.

We can conclude that the most commonly used UML diagrams for test case generation are UML state machine, activity and sequence diagrams and the class and object diagram in combination with other UML diagrams. The found approaches generate test cases with the help of some additional methods, most commonly graphs, trees, tables, labelled transition systems (LTS), genetics algorithms (GA) and finite state machines (FSM). We can also conclude that UML State Machine diagrams are most suitable for the unit testing level and UML sequence diagram for integration testing level. However, this is not the only option; we also found approaches that produce test cases for different testing levels and types. On the other hand, the UML activity diagram and approaches with more than one UML diagram are usually generating test cases for system level testing. Especially those approaches that also present an implemented testing tool. We can also conclude that not all UML diagrams are appropriate for all testing levels and types.

Despite the fact that the field of test case generation from UML models is investigated in detail, there are still some possibilities for further work. We would like to continue our work and we plan to proceed and make a more detailed analysis of SLR. We also want to implement the most promising approaches and compare them according to different properties, like effort used for test case generation, the need to supplement the model, the test case coverage of a model, the number of found mistakes, etc. Adjusted and optimized approaches could be used for testing mobile applications while taking into account different mobile characteristics and assessing and evaluating the quality of generated test cases based on different criteria, the number of found mistakes, and coverage, such as bottleneck detection and code duplication detection.

REFERENCES

- AGGARWAL, M. AND SABHARWAL, S., 2012. Test Case Generation from UML State Machine Diagram: A Survey. *Computer and Communication Technology (ICCCT), 2012 Third International Conference on*, pp.133–140.
- CARTAXO, E.G., NETO, F.G.O. AND MACHADO, P.D.L., 2007. Test case generation by means of UML sequence diagrams and labeled transition systems. *Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on*, pp.1292–1297.
- CHOUHAN, C., SHRIVASTAVA, V. AND SODHI, P.S., 2012. Test Case Generation based on Activity Diagram for Mobile Application. *International Journal of Computer Applications*, 57(23), pp.4–9.
- DALAL, S.R. ET AL., 1999. Model-based testing in practice. *Software Engineering, 1999. Proceedings of the 1999 International Conference on*, pp.285–294.
- FAN, X., SHU, J., LIU, L. AND LIANG, Q.J., 2009. Test Case Generation from UML Subactivity and Activity Diagram. *Electronic Commerce and Security, 2009. ISECS '09. Second International Symposium on*, 2, pp.244–248.
- FRANCISCO, M.A. AND CASTRO, L.M., 2012. Automatic Generation of Test Models and Properties from UML Models with OCL Constraints. In *Proceedings of the 12th Workshop on OCL and Textual Modelling*. OCL '12. New York, NY, USA: ACM, pp. 49–54.
- GULIA, P. AND CHILLAR, R.S., 2012. A New Approach to Generate and Optimize Test Cases for UML State Diagram Using Genetic Algorithm. *SIGSOFT Softw. Eng. Notes*, 37(3), pp.1–5.
- KANSOMKEAT, S., OFFUTT, J., ABDURAZIK, A. AND BALDINI, A., 2008. A Comparative Evaluation of Tests Generated from Different UML Diagrams. *Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2008. SNPD '08. Ninth ACIS International Conference on*, pp.867–872.
- KAUR, A. AND VIG, V., 2012. Systematic Review of Automatic Test Case Generation by UML Diagrams. *International Journal of Engineering Research & Technology*, 1(6).
- KHANDAI, M., ACHARYA, A.A. AND MOHAPATRA, D.P., 2011. A novel approach of test case generation for concurrent systems using UML Sequence Diagram. *Electronics Computer Technology (ICECT), 2011 3rd International Conference on*, 1, pp.157–161.
- KITCHENHAM, B. AND CHARTERS, S., 2007. *Guidelines for performing Systematic Literature Reviews in Software Engineering*.
- MINGSO, C., XIAOKANG, Q. AND XUANDONG, L., 2006. Automatic test case generation for UML activity diagrams. In *Proceedings of the 2006 international workshop on Automation of software test*. AST '06. New York, NY, USA: ACM, pp. 2–8.
- NAYAK, A. AND SAMANTA, D., 2011. Synthesis of test scenarios using UML activity diagrams. *Software & Systems Modeling*, 10(1), pp.63–89.
- OMG, 2011. *Unified Modeling Language Version 2.4.1*.
- PRETSCHNER, A. ET AL., 2005. One Evaluation of Model-based Testing and Its Automation. In *Proceedings of the 27th International Conference on Software Engineering*. ICSE '05. New York, NY, USA: ACM, pp. 392–401.
- SAMUEL, P., MALL, R. AND BOTHRA, A.K., 2008. Automatic test case generation using unified modeling language (UML) state diagrams. *Software, IET*, 2(2), pp.79–93.
- SCHWARZL, C. AND PEISCHL, B., 2010. Test Sequence Generation from Communicating UML State Charts: An Industrial Application of Symbolic Transition Systems. *Quality Software (QSIC), 2010 10th International Conference on*, pp.122–131.
- SHAMSODDIN-MOTLAGH, E., 2012. A Review of Automatic Test Cases Generation. *International Journal of Computer Applications*, 57(13), pp.25–29.
- SHIROLE, M. AND KUMAR, R., 2013. UML Behavioral Model Based Test Case Generation: A Survey. *SIGSOFT Softw. Eng. Notes*, 38(4), pp.1–13.
- SHIROLE, M., SUTHAR, A. AND KUMAR, R., 2011. Generation of Improved Test Cases from UML State Diagram Using Genetic Algorithm. In *Proceedings of the 4th India Software Engineering Conference*. ISEC '11. New York, NY, USA: ACM, pp. 125–134.
- SWAIN, S.K., PANI, S.K. AND MOHAPATRA, D.P., 2010. Model Based Object-Oriented Software Testing. *Journal of Theoretical & Applied Information Technology*, 14(1/2), p.30.
- UTTING, M. AND LEGEARD, B., 2007. *Practical Model-Based Testing: A Tools Approach*, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.
- WANG, X., GUO, L. AND MIAO, H., 2008. An Approach to Transforming UML Model to FSM Model for Automatic Testing. *Computer Science and Software Engineering, 2008 International Conference on*, 2, pp.251–254.
- ZENG, F., CHEN, Z., CAO, Q. AND MAO, L., 2009. Research on Method of Object-Oriented Test Cases Generation Based on UML and LTS. *Information Science and Engineering (ICISE), 2009 1st International Conference on*, pp.5055–5058.

Prototype System for Improving Manually Collected Data Quality

JARI SOINI, PEKKA SILLBERG and PETRI RANTANEN, Tampere University of Technology – Pori

Even nowadays, a great deal of measurement data is collected and also saved manually. In this kind of situation, there are phases when human error can easily occur and also when interpreting the typed collected measurement data could be difficult. This research aimed to discover resources for improving the quality of measurement data as well as better and more illustrative tracking of usage information in real time. The objective was both quality improvement of a specific measurement data collection process as well as the elimination of human error. This paper describes one reliable solution for this purpose, which improves the quality and also the visual presentation of manually collected data. The paper presents elements of the system developed for this aim and also the technology deployed along with its operational principles.

Categories and Subject Descriptors: H.3.5 [Information Storage and Retrieval]: Online Information Services—*Web-based services*; H.4.0 [Information Systems Applications] General

Additional Key Words and Phrases: Data quality, measurement process quality, data visualization, software applications

1. INTRODUCTION

The starting point of the research was to map out areas and activities of the public sector in which *savings* could be achieved by controlling, optimizing and intensifying operations. This research is a part of the ongoing two-year (2013-2014) Kiiaudata (Kiinteistöjärjestelmien datan älykäs analysointi – smart analysis of property systems data) project funded by Tekes [2014], where one of the main aims was to study potential new technologies for managing and controlling conditions in buildings in a smart way. In collaboration with the City of Pori, a survey was made about the points where measurement data is collected and also how said data is utilized. As the result of this mapping, it was decided to focus on the upgrading of measurement data collection and the new swimming pool was chosen as the research subject, as it is the city's most expensive individual building in terms of energy consumption.

The idea was that the maintenance staff would continue checking the physical measuring devices to ensure their conditions, but the collected data would be recorded with the developed system in contrast to the fully manual record keeping used in the past (i.e. pen and paper). The measurements produce information that can be used, for example, in consumption and condition tracking. For instance, analyses of alteration in energy consumption can be made by means of inclusive measurement and usage tracking based on it. Electricity, heat and water are examples for different measured energy currents. In many cases, the aforementioned currents can be tracked and anomalous situations can be reported automatically using modern computer controlled systems, but there still remain situations where manual work is required, especially when dealing with legacy systems.

Author's address: J. Soini, P. Sillberg and P. Rantanen, Department of Software Engineering, Tampere University of Technology – Pori, P.O.Box 300, FIN-28101 Pori, Finland; email: {jari.o.soini, pekka.sillberg, petri.rantanen}@tut.fi

Copyright © by the paper's authors. Copying permitted only for private and academic purposes.

In: Z. Budimac, T. Galinac Grbac (eds.): Proceedings of the 3rd Workshop on Software Quality, Analysis, Monitoring, Improvement, and Applications (SQAMIA), Lovran, Croatia, 19.-22.9.2014, published at <http://ceur-ws.org>.

There are several studies related to building automation systems and automatic sensor data collection, for example Cheng and Shen [2011] introduced wireless sensor networks based on embedded Linux. Nainwal et al. [2011] studied on remote surveillance and monitoring system utilizing wireless sensor networks, Vujović and Maksimović [2014] focused on utilizing Raspberry Pi as a building block of wireless sensor node, and Toshniwal and Conrad [2010] introduced a web-based sensor monitoring system on a Linux-based single board computer platform. However our focus was on systems where automatic sensors cannot be fully utilized. The work presented in this paper utilizes the findings of Soini et al. [2013], in which mobile devices, Global Positioning System (GPS) technology and route optimizations were combined in a real-time tracking service for delivery of goods.

The owners of the property chosen as the research subject – the new public indoor swimming pool of the City of Pori – were particularly interested in, for example, identifying development targets related to energy consumption measurement, development of the measurement process, early discovery of possible issues, and evaluation of the impacts of changes. For this research, a manually used digital data collection system has been developed as a collaboration project between Tampere University of Technology (TUT) and the City of Pori. The system developed facilitates the maintenance staff's work in registering and recording the measurement information as well as real-time tracking of usage information and perception of possible anomalous consumption situations.

2. PROBLEMS IN QUALITY OF MANUALLY COLLECTED DATA

Erroneous values are common when collecting and typing up data by hand, especially for long numeric values. Errors can also be very hard to detect, and it is difficult to know if the erroneous value was caused by an error with a meter or a correct value was simply mistyped by the person reading the meter. This was the problem observed and the starting point of this study. The assumption was that typing errors can be detected by software.

In some cases, it is not financially viable to replace measuring devices: many devices available today can be networked and contain automatic error detection or monitoring software, but this is not true for all devices, especially when taking into consideration many legacy devices. If these devices are seldom used or replacing them would be expensive, alternative approaches are required.

There are still many measuring devices that need to be checked periodically by a user. In practice this may require writing down the values by hand. In many places it is still common to use the basic pen-paper-and-Excel approach, in which the measurements are checked manually, written down and later inputted using datasheet software such as Microsoft Excel. The system presented here enables the pen-and-paper phase to be skipped. Using a management interface, reports of the values can be created and saved in various formats (such as .pdf or .xls). The paper describes simple client software, which uses Near Field Communication (NFC) [ISO 2013] tags to detect a measurement device called an “object” in the context of this paper. In the scope of this paper, an object means a monitored physical device (e.g. water meter).

3. SOLUTION – PROTOTYPE SYSTEM FOR COLLECTION OF CONSUMPTION DATA

The main idea behind the prototype system is to combine a typical web service, a mobile device with networking capability and a way to identify the object to implement a data gathering and reporting service. QR-Codes and Radio Frequency IDentification (RFID) [ISO 2008; Finkenzeller 2010] contactless proximity cards were the main candidates for identification purposes. RFID cards were chosen over QR-Codes as they should be more reliable to recognize in dimly lighted environments. It is also more convenient to touch the card instead of taking a photo of QR-Code when the space is limited.

Not all RFID cards, or tags, are alike as they vary on parameters such as operating frequency, data speed, distance of reading, power supply (passive, active, battery-assisted passive), and price. The

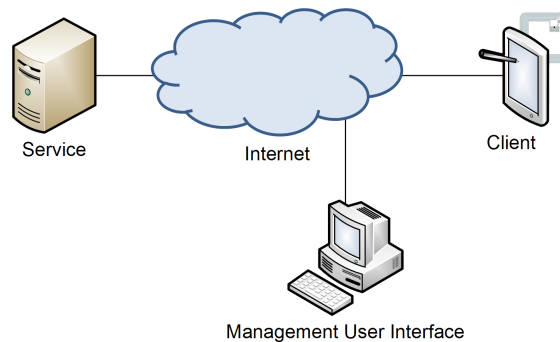


Fig. 1. System overview.

choice of parameters depends on the use case [Nummela 2010]. Typically, a low operating frequency correlates with low data speed and reading distance. An active power supply increases the price of the tag but enables the tag to operate without the support of a tag reader. We chose to use Near Field Communication (NFC) compatible tags as they are:

- relatively inexpensive
- they receive all the required power from the reader which reduces the need for maintenance
- the reading distance was not a crucial part of the system
- NFC capable smart phones and tablets are becoming more common.

For the purpose of this application, we are only interested in the unique ID which can be read from every tag. In our system this ID – i.e. a tag – is bound to an object. The user only has to touch the tag and the client software retrieves the correct data. Every object can be configured with various details:

- a common name (e.g. Water consumption)
- names of related gauges (e.g. Main water meter)
- the unit of the gauge (e.g. Cubic meters)
- warning limits for expected minimum and maximum daily increase (e.g. we expect that the gauge reading could increase by 50 to 100 units per day).

Figure 1 shows an overview of the system. The *Service* is available over the Internet where both *Management User Interface* and *Client* application can be connected. The service uses JavaScript Object Notation (JSON) to transmit data objects and it has two Representational state transfer (REST) interfaces, one for getting the gauge data and the other for posting the gauge data. It also supports user access control, but this feature is not currently used in the pilot phase of the system. The management user interface is a JavaScript-based web page accessible with a web browser. There the system administrator can configure a particular object and interpret the results sent by the client. For example, the results can be viewed as raw data or plotted as a chart. The *Client*, in Figure 1, is the main component that the end user is using. It is used to interact with tags, collect the data, and perform small scale on-site analysis of the data. The client application in our case is programmed for Android devices.

The prototype system is currently being tested in at the new swimming pool in City of Pori. There are three different gauges (water, electricity and central heating) which are being monitored. There are a couple members of the maintenance staff who operate the client device just to collect the gauge readings, and one person to oversee the changes in the collected data. All workers are operating the

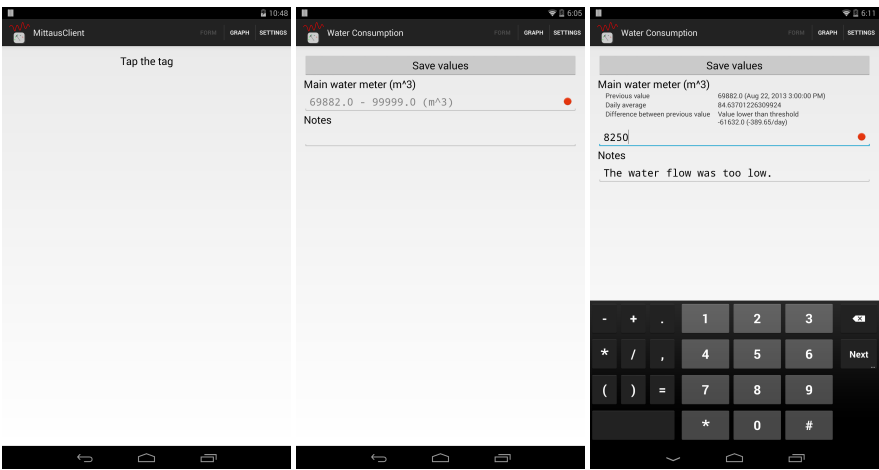


Fig. 2. Application screenshots, from left to right: initial view before a tag has been read, form view after the tag has been read, and finally, threshold value is below the defined limit.

client thru one shared device. So far the response from the staff has been enthusiastic about the data collection system, particularly of the ability to see the approximate costs of the facility immediately.

3.1 Information Collection

The client device and the NFC tag play an important role in collecting the meter readings. The information collection consists of three phases:

- (1) identifying the object
- (2) inputting the data
- (3) saving the data.

Each of the three phases is explained in more detail in the following sub-sections.

3.1.1 Identifying the Object. The first step is to identify the object by touching the tag attached to the object. The tag will be automatically detected by the device. The tag detection is based on the unique ID found on every tag. In the current implementation these IDs can be mapped to objects using the service's management interface. This mapping is used by the client to detect which object is the current target and to show the correct object-dependent input fields. It could also be possible to extend the client software to enable mapping new tags for objects, which would make installing the overall system easier. This way the system installation could use bulk tags, which would be mapped to objects on the spot by the person performing the installation procedure. Whether mapping the tags on the device is required depends on the use case, and in our current scenario it was not a necessary feature mainly because of the relatively small amount of objects and tags. Also, as the main use of the client device is to gather information, it might be better to keep the software simpler to use by limiting the functionality available (see Figure 2).

The mapping information and the input field details can be synchronized with the service at any time, but in general, synchronization is performed only when specially requested. There are two reasons for this: firstly, the mapping and input field details change very rarely, making continuous synchronization a waste of network bandwidth; and secondly, in some cases the objects may be located in places with poor or non-existent network connectivity, making live synchronization difficult or even

impossible. The basic view before any tags have been detected is shown in Figure 2 (left), and the view after a tag has been selected is shown in Figure 2 (center). In the example case a very simple object is illustrated containing only two fields; a numerical input field for the *Main water meter*, which accepts values ranging from 69882 (the previous input value) to 99999, and a text input field for *Notes*.

3.1.2 Inputting the Data. Figure 2 shows the views of a detected object. The view in the center shows the basic view and the view on the right show the extended view. When the user taps any of the fields, additional information related to that specific field is shown: the previously given value with the timestamp of the input date, the daily average, and the difference of the currently typed value (if any) in relation to the previously given value. The purpose of the extended information is to give a quick glimpse of previous data, which can be used to detect possible errors in the readings and give the person using the device an idea of the possible values. In the example case (Figure 2, right), the red dot on the right hand side of the input field shows that a bad value has been given, and the user has typed a descriptive comment on the matter in the *Notes* section (“The water flow was too low”). Figure 3 (left side) shows the same case with the properly inputted value.

The value ranges used to detect and show warning situations are configured on the management web interface of the service. The ranges are numerical thresholds, which have either been calculated based on earlier data (e.g. it may be known how much water is used on average on a daily basis), or they may be based on physical limits (e.g. water consumption cannot be negative). The ranges can be simple minimum and maximum values, which should not be exceeded (e.g. voltage should stay between 10 and 15 volts) or cumulative limits (e.g. water consumption should not exceed ten cubic meters per day). The minimum and maximum values do not need previous values for accurate calculation of the warning threshold. In the case of cumulative limits, at least one single previous value is required. The previous values can be provided by the service when synchronizing the tag mappings and input fields or they can be results from previous use of the software. The warnings are meant to help the person typing the input values, and they are only “soft limits”, i.e. they can be overridden if required. For example, it may be possible that a meter is giving an erroneous reading or for some reason much higher consumption is occurring. In this case it may be required to input a value that is outside the previously designated range. Inputting a value outside the range requires a confirmation from the user, and it will automatically be detected by the system and will pop up as an erroneous value on the management interface. It is also possible to generate an automatic notification, for example an email or SMS alert to be sent when an erroneous value is detected by the service, but in practice the notification will not be sent immediately if the data inputting process is performed in a location without network connectivity.

3.1.3 Saving the Data. After the user has inputted the desired data, the *Save values* button can be used to submit the results. The submit process may not necessarily start immediately. The values are stored locally on the device and can be viewed at any time, but when the actual result submission happens depends on the availability of the service. The client contains a background service which will periodically try to submit any unsent information. In our use case, the measuring devices themselves are located in an area of poor connectivity, but the users’ workplace contains areas where the results can be submitted. The users generally carry the client device with them, thus allowing the automatic submission of the results when a network connection has been established. If instantaneous submission is required, other approaches should be considered such as providing wireless access by using a wireless router. The effects of periodic submit retries on battery life may vary. On one hand, turning the wireless radio off, and turning it on only when required may improve the battery life of the device. On the other hand, if the availability of the network connectivity is unknown, it may be difficult to establish the connection at timed intervals. In practice, many tablet and smart phone devices can

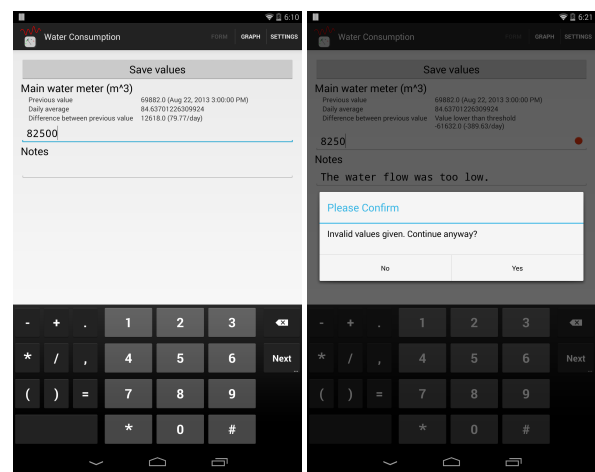


Fig. 3. Application screenshots: the left figure is the form view with data given by the user, the right figure asks for user confirmation before saving the data.

sustain a battery life of a whole day using the default power saving settings, thus only requiring the device to be charged when not needed, for example, outside working hours.

If the inputted values contain erroneous out-of-range values, a confirmation of values is required before the data can be saved and sent. The confirmation dialog is illustrated in Figure 3 (right).

3.2 Viewing the Results

The system allows the user to examine the collected data quickly on the client application and more thoroughly using the management user interface. Figure 4 illustrates the general idea of the different views:

- simple chart view of the client application on the left
- more complete analysis chart of the management user interface on the right.

The rationale for limiting client application features is to keep it as simple as possible and therefore to reduce the maintenance required for the application. It also helps to keep the device small enough for carrying around and for entering data. Also, the employee typing in the data might be more interested in seeing if the figures show any unexpected highs or lows, so he/she can react to the situation more quickly.

Both charts in Figure 4 contain the same data (consumption of water; x-axis time; y-axis consumption in cubic meters), but the view on the client application (Figure 4, left) is panned and zoomed in to show data between June 2013 and August 2013. The browser view (Figure 4, right) displays all of the data beginning from January 2012 and ending in May 2014. The upper chart shows the actual data and the lower chart illustrates the calculated daily average consumption. Between the charts there is a section with statistical information about the consumption. It shows the meter reading, date of the reading, and also approximated daily, weekly, and monthly costs in euros (by using a predefined average price per unit). The statistical information follows the pointer of the mouse so it is possible to see the same data from any point of the chart.

A surge in water consumption can be seen during July 2013 with consumption peaking at 400 cubic meters per day. This kind of information can be helpful for the maintenance team as it could be a sign of a leakage somewhere in the system. Fortunately, the peak was due to a scheduled maintenance of

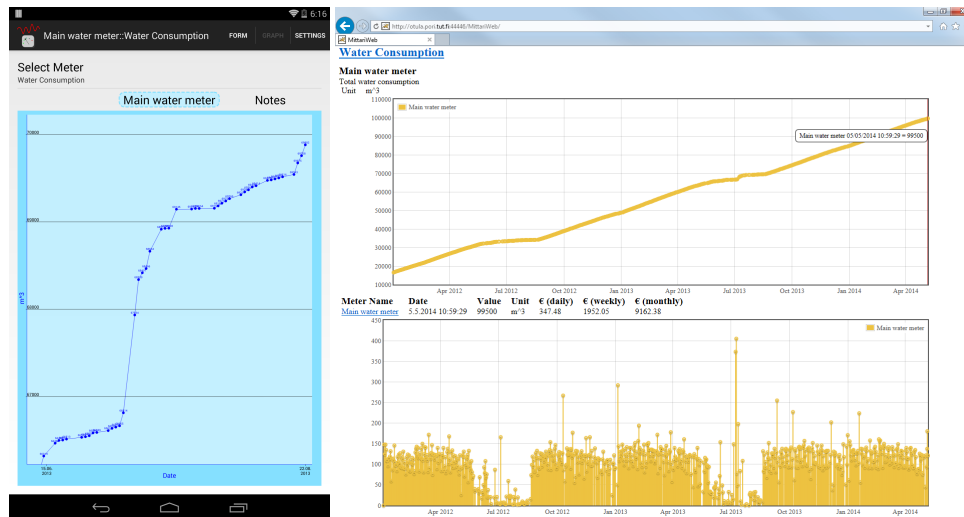


Fig. 4. Chart views as seen on mobile application (left) and on web browser (right).

the swimming pools. There are also many small consumption peaks and lows on the lower chart of the browser view. This occurred because the data was imported from the handwritten notes without exact time information. The collection time of the imported data is simply set at 12 noon, so it will cause fluctuation if the meter was actually read in the morning or evening. In the future as the data is collected directly to the system, the exact reading time can be stored, which will eliminate the fluctuation caused by unknown meter reading times.

The data shown in Figure 4 has been imported into the system from the actual water consumption data collected from the new public swimming pool located in the City of Pori. The facility has also been recording the consumption of central heating and the consumption of electrical energy. As the data comes from an actual facility, we had the opportunity to reflect on the consumption in terms of what had really happened. The data can be broken down into the following sections (see Figure 4, right side):

- (1) January 2012 – June 2012, (winter & spring season, average consumption)
- (2) June 2012 – August 2012 (summer maintenance, low consumption)
- (3) August 2012 – June 2013 (fall, winter & spring season, average consumption)
- (4) June 2013 – August 2013 (summer maintenance, from low to high consumption)
—Contains a surge of water consumption due to pools being emptied, overhauled, and then refilled.
- (5) August 2013 – May 2014 (fall, winter & spring season, average consumption)

The data has been recorded by pen-and-paper, but is now being stored directly on the electronic database by using the system described in this paper. In fact, there are a lot of other digitally monitored and configurable parameters in the new swimming pool facility, but these three gauges (water consumption, central heating consumption and electrical energy consumption) are the only meters that still require old-fashioned manual reading.

4. DISCUSSION

The efficiency of the system greatly depends on the defined value ranges. If it is not possible to define clear ranges or the ranges remain vague, the possibility of error increases, and in this case the software works only as a pen-paper-and-Excel replacement. In practice, based on user feedback, the most

common source of error was grossly mistyped numbers, caused by lengthy numeric values (e.g. when writing down values it is easy to mix up 154763 and 157463, an error that can easily be detected by the software).

The software is more suitable for use cases where the meters are not read very often, but *do need* to be read manually periodically. If the meters need to be read continuously, for example several times a day, it may be more advisable to invest in meters with an automatic monitoring and warning system (if possible). On the other hand, if the meters are hardly ever checked, the basic pen-paper-and-Excel approach may be more feasible, and the resources required for setting up the system can be saved.

Then why not change the remaining analog meters? The comments from the facility's maintenance workers were that if they routinely read the meters every day, they can simultaneously monitor the condition of the nearby equipment and perform preventative maintenance if needed. Thus they can complete several tasks at once. It also helps to get a better grasp of the facility as a whole as they can see how much power or water is consumed daily.

5. SUMMARY

The paper presents a system for improving the quality of manually collected data. In many cases, especially in the public sector, there are many different points where manually measurement data collection is still practised. These situations usually relate to the monitoring of the operations of some physical devices, such as energy-related consumption measurement. The system introduced assists maintenance staff and also supports managers who are responsible for ensuring the correct operation of the devices. This system is one step towards more reliable and thus better quality measurement data, and it also improves the visual presentation of collected data for analysis. During the ongoing study, the system features will be extended and adapted for the purpose of monitoring patient rooms in the public sector health care environment.

REFERENCES

- Xiaohui Cheng and Fanfan Shen. 2011. Design of the wireless sensor network communication terminal based on embedded Linux. *2011 IEEE 2nd International Conference on Software Engineering and Service Science* (July 2011), 598–601.
- Klaus Finkenzeller. 2010. *RFID Handbook: Fundamentals and Applications in Contactless Smart Cards, Radio Frequency Identification and Near-Field Communication* (3rd ed.). Wiley.
- ISO. 2008. *ISO/IEC 14443, Identification cards – Contactless integrated circuit cards – Proximity cards*.
- ISO. 2013. *ISO/IEC 18092:2013, Information technology – Telecommunications and information exchange between systems – Near Field Communication – Interface and Protocol (NFCIP-1)*.
- V Nainwal, P J Pramod, and S V Srikanth. 2011. Design and implementation of a remote surveillance and monitoring system using Wireless Sensor Networks. In *Electronics Computer Technology (ICECT), 2011 3rd International Conference on*, Vol. 5. 186–189.
- Jussi Nummela. 2010. *Studies towards Utilizing Passive UHF RFID Technology in Paper Reel Supply Chains*. Doctoral dissertation. Tampere University of Technology.
- Jari Soini, Timo Widbom, Jari Leppäniemi, Petri Rantanen, and Pekka Sillberg. 2013. Pilot system for transport confirmation with location awareness. In *Symposium GIS Ostrava 2013 - Geoinformatics for City Transformation*. Ostrava.
- Teke. 2014. Finnish Funding Agency for Technology and Innovation. (2014). <http://www.tekes.fi/eng>
- Kailash Toshniwal and James M. Conrad. 2010. A web-based sensor monitoring system on a Linux-based single board computer platform. *Proceedings of the IEEE SoutheastCon 2010 (SoutheastCon)* (March 2010), 371–374.
- Vladimir Vujović and Mirjana Maksimović. 2014. Raspberry Pi as a Wireless Sensor Node : Performances and Constraints. In *Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2014 37th International Convention on*. Opatija, 1247–1252.

The Effect of Educating Users on Passwords: a Preliminary Study

VIKTOR TANESKI, BOŠTJAN BRUMEN and MARJAN HERIČKO, University of Maribor

Passwords are a basic authentication method for most information systems. Despite their widespread use, passwords still suffer from a number of problems. Users and their passwords are the Achille's heel (the weakest link) of security, because they still tend to create passwords that are weak, easy to remember and contain words that are familiar to them. They also tend to trade security for memorability. Users' lack of security consciousness and their behaviour can be influenced by information security training. This paper presents the preliminary results of our research in progress. Our research explores the effect of password security training on strength of the passwords chosen by the users and their consciousness about security and the importance of creating strong and hard-to-guess passwords. We collected the data by means of an online questionnaire, performed among undergraduate students from the Faculty of electrical engineering and computer science at the University of Maribor. Overall, the results show that, despite our lectures and recommendations, users still lack of security knowledge regarding password change and password write-down and need to be further educated in this direction.

Categories and Subject Descriptors: K.3.2 [**Computers And Education**]: Computer and Information Science Education—Accreditation, Computer science education, Curriculum, Information systems education, Literacy, Self-assessment; K.4.0 [**Computers and Society**] General; K.6.5 [**Management Of Computing And Information Systems**]: Security and Protection—Authentication, Unauthorized access

General Terms: Human Factors, Security

Additional Key Words and Phrases: Authentication, computer security, passwords, password security, password security education

1. INTRODUCTION

Password-based authentication mechanisms are the primary means by which users gain legitimate access to a computer system. Although alternative authentication methods such as biometrics (based on “what you are”), smart cards (based on “what you have”) and two-step verification are becoming more available, passwords remain the most common method for authentication for computer systems. But passwords suffer from a number of problems known for some time now. Password-related problems are firstly identified by Morris and Thompson [1979]. The authors conducted experiments in order to determine typical users' habits about the choice of passwords and noticed that users of the system chose passwords that are short, simple and contained only lowercase letters and digits, or appeared in various dictionaries. These findings were confirmed 20 years later by Zviran and Haga [1999]. The authors concluded that one of the biggest vulnerabilities to a computer system's security is the user. Almost 50

Author's address: V. Taneski, Faculty of electrical engineering and computer science, University of Maribor, Smetanova ul. 17, 2000 Maribor, Slovenia; email: viktor.taneski@um.si; B. Brumen, Faculty of electrical engineering and computer science, University of Maribor, Smetanova ul. 17, 2000 Maribor, Slovenia; email: bostjan.brumen@um.si; M. Heričko, Faculty of electrical engineering and computer science, University of Maribor, Smetanova ul. 17, 2000 Maribor, Slovenia; email: marjan.hericko@um.si.

Copyright © by the paper's authors. Copying permitted only for private and academic purposes.

In: Z. Budimac, T. Galinac Grbac (eds.): Proceedings of the 3rd Workshop on Software Quality, Analysis, Monitoring, Improvement, and Applications (SQAMIA), Lovran, Croatia, 19.-22.9.2014, published at <http://ceur-ws.org>.

percent of the users surveyed in the study reported passwords composed of five or fewer characters, 80 percent used only alphanumeric characters, and 80 percent never changed their password.

A recent literature review in the area of textual passwords and textual passwords security Taneski et al. [2014], summarises the most common problems related to creating and managing textual passwords. The review also summarises proposed solutions and approaches for better coping with the identified problems with textual passwords. Many approaches and solutions, for increasing the security and usability of textual passwords, were proposed through the years but none have proven widely acceptable. The review shows that, despite all these approaches, recommendations, security policies and etc., users still encounter the majority of the identified and already known problems. Users and their textual passwords are still considered “the weakest link”, because they still tend to choose weak passwords and passwords that can be found in a dictionary [Egelman et al. 2013] and are likely to use words that are familiar to them as their passwords [Bishop and Klein 1995]. Because of the rapid growth of the popularity of Internet technology and the increased number of online services requiring password-based authentication, users have to maintain many different accounts and have to remember multiple passwords. This leads to users to frequently forget their passwords, write them down or share them with their friends [Jakobsson and Dhiman 2013]. Users create the easiest-to-remember passwords regardless to any recommendations or instructions and tend to trade security for memorability [Zviran and Haga 1990; Zviran and Haga J. 1993]. These problems can arise as a result of users’ lack of security motivation and understanding of password policies and the fact that users tend to circumvent password restrictions for the sake of convenience [Adams et al. 1997]. This indicates a deficit in user’s consciousness about security and this user behaviour can be influenced by information security training [Horchler and Tejay 2009]. Educating users about password security and the importance of creating strong and hard-to-guess passwords and assisting them with creating secure passwords can raise the security consciousness of system users and can help achieve greater security [Zviran and Haga 1999].

This study, which is research in progress, explores the recommendation of Zviran and Haga [1999] by analysing the effect of password security training on user’s practices regarding password creation, password use and management and their consciousness about security and the importance of creating strong and hard to guess passwords. We performed a pilot study on university students about their password practices, attitudes and knowledge about password creation and password security. We administered the survey in two phases over one month period along with several lessons on password security between the two phases. The first phase of the survey was performed among students that had no password security training. The second phase of the survey was performed one month later on the same group of students. For a period of two weeks, between the two phases, the students were educated about the importance of password security and how to manage their passwords. Following the lectures there was a two week period for learning decay. The contribution of this study in this manner is that we succeeded in educating the participants about some password practices, in contrast to some previous studies Hart [2008], Dhamija and Perrig [2000] that have not proven to be successful at all. Finally, we present the preliminary results of our survey.

The rest of the paper is organized as follows: in Section 2 we describe the research method; in Section 3, we present the results of the survey and a discussion; Section 4 concludes the paper and presents plans for future work.

Table I. Questionnaire Categories

Category	Description
Account information	which services or devices does the student have a password for (desktop computer, notebook computer, mobile phone, tablet, University account, Facebook account, Twitter account, Google account)
Password information	the characteristics of the selected passwords (first character, length, how did the user choose the passwords etc.)
Password recall	how often do users forget their passwords, how often do they change them and how often do they write their passwords down
Frequency of password use	information about the frequency of password use (how often do users log in by using the password, and whether they use the password to log in to multiple accounts)
Data importance and data sensitivity	how important and how sensitive is to users the data that they are protecting
Demographic data about the user	gender, year of birth, university, faculty, postal code etc.

2. METHOD

2.1 The Survey

Data for this study is collected by means of an online questionnaire. The main objective of the questionnaire is to determine the characteristics of textual passwords used by individuals to log in, mainly to a university account, but also to other accounts, such as Facebook, Twitter, Google, or their own mobile devices and phones. Because asking users about their passwords is a sensitive topic, we did not ask about their actual passwords but about the characteristics of their passwords for different accounts. We classify the questions into six categories, presented in Table I.

For creating our questionnaire, we drew inspiration from the questionnaire that was performed by Zviran and Haga [1999]. Our questionnaire has additional questions and additional answer options about password characteristics for different accounts for different services or devices, thus allowing us to compare characteristics of different passwords for different accounts. The completion of the questionnaire took between 15 and 20 minutes.

2.2 Participants

A total of 33 undergraduate students from the Faculty of electrical engineering and computer science at the University of Maribor, participated and completed the first phase of the web-based survey. 30 of them also completed the second phase of the study, but 24 attended the lectures about password security. All of the participants were regular users of the Internet and modern mobile devices and smart phones, with one or more different password-protected accounts. Five participants were female and 28 male in the first phase, and three were female and 27 male in the second phase. The average age of the participants is 22.

2.3 Procedure

The survey was conducted in spring term 2014 and consisted of two phases. The first phase of the questionnaire was performed among students that did not receive the lessons. After the first phase of the survey, the students attended lectures dedicated specifically to passwords and related issues, like password creation, management and security. The education consisted of topics about the importance of creating strong and secure passwords, how to choose such passwords and how to manage them. After the education part followed a two week period of learning decay, followed by the second phase of the survey. We slightly modified the questionnaire by adding an additional answer option “I prefer not to disclose” to every question in the “Password information” section. By adding this answer option, we want to determine if the users that attended the lectures are more conscious about the importance of concealing information about their passwords.

Table II. Summary of Common Password Characteristics (Phase 1)

Account	Common password characteristics	Count	No answer
desktop computer	alphabetic - lowercase letters only (e.g. password)	5 (15.15%)	17 (51.52%)
notebook computer	alphabetic - lowercase letters only (e.g. password)	7 (21.21%)	4 (12.12%)
mobile phone	numeric - digits only (e.g. 12345)	13 (39.39%)	11 (33.33%)
tablet device	numeric - digits only (e.g. 12345)	2 (6.06%)	28 (84.85%)
University account	alphanumeric - single case letters and 1-2 digits behind (e.g. pass1, pass12, PASS1, PASS12)	9 (27.27%)	3 (9.09%)
Facebook account	alphanumeric - mixed case letters and 3 or more digits (e.g. Pass1W34oR9d)	7 (21.21%)	3 (9.09%)
Twitter account	alphanumeric - mixed case letters and 3 or more digits (e.g. Pass1W34oR9d)	4 (12.12%)	21 (63.64%)
Google account	alphanumeric - mixed case letters and 3 or more digits (e.g. Pass1W34oR9d)	7 (21.21%)	6 (18.18%)

Table III. Summary of Common Password Characteristics (Phase 2)

Account	Common password characteristics	Count	No answer
desktop computer	alphanumeric - mixed case letters and 1-2 somewhere (e.g. Pass12Word, Pa12ssWord, 1PassWord2)	5 (20.83%)	9 (37.50%)
notebook computer	alphanumeric - mixed case letters and 1-2 somewhere (e.g. Pass12Word, Pa12ssWord, 1PassWord2)	7 (29.17%)	2 (8.33%)
mobile phone	numeric - digits only (e.g. 12345)	9 (37.50%)	9 (37.50%)
tablet device	numeric - digits only (e.g. 12345)	1 (4.17%)	22 (91.67%)
	alphanumeric - single case letters and 1-2 digits in between (e.g. pass12word, PASS12WORD, P12ASSWORD)	1 (4.17%)	
University account	alphanumeric - single case letters and 1-2 digits behind (e.g. pass1, pass12, PASS1, PASS12)	5 (20.83%)	2 (8.33%)
Facebook account	alphanumeric - mixed case letters and 3 or more digits (e.g. Pass1W34oR9d)	4 (16.67%)	2 (8.33%)
	mixed case letters, several special characters and several digits (e.g. 3Pa!s45Wor\$d)	4 (16.67%)	
Twitter account	alphanumeric - mixed case letters and 1-2 somewhere (e.g. Pass12Word, Pa12ssWord, 1PassWord2)	2 (8.33%)	16 (66.67%)
	alphanumeric - single case letters and 3 or more digits (e.g. 1PASS23WORD4, pass1wor3d4)	2 (8.33%)	
Google account	alphanumeric - mixed case letters and 1-2 somewhere (e.g. Pass12Word, Pa12ssWord, 1PassWord2)	4 (16.67%)	5 (20.83%)

3. RESULTS

We compared password characteristics of the user group before the users attended the password security lectures (phase 1) and after the lectures (phase 2). Thirty participants attended the second phase of the questionnaire, but not all of them (24) attended the lectures. We took in consideration only the participants that attended the lectures. In this section, we summarise the preliminary results of our study, by comparing the results from the two phases of the survey. We present the results about the general password characteristics (password length, password composition), password change frequency, and password memorability and write-down.

3.1 General Password Characteristics

While we ask our participants about their password characteristics, we do not inquire their actual passwords. The question that we ask in both phases is “What are the characteristics of your password?”. The answers to the question for both phases are summarised in Table II and Table III respectively. In both tables, the first column denotes the specific accounts, the second column represents the most common answers about password characteristics for every account, the third column represents the percentage of users who chose that answer. The last column of each table represents the number of

Table IV. Average Password Length

Account	Average password length		
	Phase 1	Phase 2	Not disclosing
desktop computer	8.81	10.77	2
notebook computer	8.79	11.00	4
mobile phone	4.91	6.08	2
tablet device	5.40	5.50	0
University account	7.67	8.70	2
Facebook account	10.33	12.06	4
Twitter account	9.00	10.33	2
Google account	10.48	12.59	2

participants that did not answer the question for the specific account (since there is a possibility that the user had not created that account). The results summarised in Table II show that the passwords for the desktop and notebook account consist of most commonly lowercase letters only, regardless the fact that 15.15% of the users rated their data on their desktop computer as very important, and 45.45% rated their data on their notebook computer, also as very important. The results in Table III demonstrate that the passwords for these accounts in the second phase of the study are mostly alphanumeric with mixed case letters and numbers. Thus, we can observe an improvement in the characteristics of the passwords for the desktop and notebook accounts. It is most likely that, after attending the lectures about passwords and password security, the awareness of users about the importance of their data and the importance of creating strong and hard to guess passwords for the data, increased. These findings are in line with the statement of Adams et al. [1997] that, users lack of security knowledge. Very often users are not conscious about the security and the importance of their data, and they need additional guidance on information importance and sensitivity.

In regard to the average password length, we provided lectures for the participants and gave them recommendations about the design, management and protection of strong and hard-to-guess passwords. The data about the average password length for both phases is summarised in Table IV. The first column denotes again the specific accounts, while the second and third columns represent the average password length for the first and second phase, respectively. The last column represents the number of participants in the second phase, that answered the specific question about the password length for their accounts with “I prefer not to disclose”. The results revealed an increased average password length in almost every account in the second phase of the study. One possible explanation for these findings may be that the increase of the average password length is a result of the users changing their passwords after attending the lectures.

3.2 Password Change Frequency

We trained the participants about the importance of frequent password change in order to ensure that a stolen password can not be used to compromise other passwords and accounts of other users in the system. The frequent password change is a basic security measure [Zviran and Haga 1999]. But, because of the rapid growth of the popularity of the Internet and the increased number of on-line accounts, a user has to maintain many different passwords [Notoatmodjo and Thomborson 2009]. Because of this, the forced and too frequent password changes can have negative effect on the users (users may quickly forget which password is current, which may lead to users tempting to write their passwords down or to reuse an old one) [Sasse et al. 2001]. The research in literature still supports frequent password changing to reduce predictability and suggests that a realistic time for password change for the average user may be 90-120 days [P. Cisar and Cisar 2007]. In the first phase, we asked our participants “How often do you change your password (when not required by the system)?”, and in

Table V. Frequency of Password Change (Phase 1)

Frequency	desktop comp.	notebook comp.	mobile phone	tablet device	University account	Facebook account	Twitter account	Google account
never changed it since first use	6 (18.18%)	7 (21.21%)	8 (24.24%)	2 (6.06%)	18 (54.55%)	6 (18.18%)	6 (18.18%)	7 (21.21%)
less than once a year	6 (18.18%)	12 (36.36%)	9 (27.27%)	2 (6.06%)	8 (24.24%)	12 (36.36%)	3 (9.09%)	8 (24.24%)
up to three times a year	3 (9.09%)	6 (18.18%)	2 (6.06%)	0 (0%)	2 (6.06%)	6 (18.18%)	1 (3.03%)	7 (21.21%)
four to six times a year	1 (3.03%)	3 (9.09%)	3 (9.09%)	1 (3.03%)	1 (3.03%)	5 (15.15%)	0 (0%)	3 (9.09%)
about once every month	0 (0%)	0 (0%)	1 (3.03%)	0 (0%)	0 (0%)	0 (0%)	1 (3.03%)	2 (6.06%)
more than once a month	0 (0%)	1 (3.03%)	0 (0%)	0 (0%)	1 (3.03%)	1 (3.03%)	1 (3.03%)	0 (0%)
No answer	17 (51.52%)	4 (12.12%)	10 (30.30%)	28 (84.85%)	3 (9.09%)	3 (9.09%)	21 (63.64%)	6 (18.18%)

Table VI. Password Write-down

Answer	Phase	desktop comp.	notebook comp.	mobile phone	tablet device	University account	Facebook account	Twitter account	Google account
Yes	First	2 (6.06%)	3 (9.09%)	2 (6.06%)	1 (3.03%)	6 (18.18%)	2 (6.06%)	1 (3.03%)	3 (9.09%)
	Second	1 (4.17%)	1 (4.17%)	1 (4.17%)	0 (0%)	5 (20.83%)	3 (12.50%)	0 (0%)	3 (12.50%)
No	First	13 (39.39%)	25 (76.76%)	20 (60.61%)	3 (9.09%)	23 (69.70%)	27 (81.82%)	10 (30.30%)	23 (69.70%)
	Second	12 (50%)	19 (79.17%)	13 (54.17%)	2 (8.33%)	16 (66.67%)	18 (75%)	8 (33.33%)	15 (62.50%)

the second phase, following the lectures, we asked them “whether they changed their password since the last survey or not, and why?”. Table V summarises the result of the frequency of password change in the first phase of the study. We found that many users never change their password for the specific account since its first use, or rarely change it (less than once a year). Despite our lectures and recommendations, the results show that a large percent of the users (41.67% for the Facebook account, 37.50% for the Google account, 50% for the notebook computer and even 66.67% for the University account) did not change their password since the first phase of the study. Overwhelming and promising are the findings about the users who changed their passwords since the first phase and after attending the lectures. Even 25% for the desktop computer, 33.33% for the notebook computer, 20.83% for the University account, 29.17% for the Facebook account, and 25% for the Google account, stated that they changed their passwords because they realized that the old password was very weak and could compromise their other passwords and accounts.

3.3 Password Memorability and Write-down

Part of our lectures and recommendations consisted of educating the participants about how important it is for them not to write their passwords down on any item (notebook, on a desk, keyboard, monitor, wall, in their mobile phones etc.). Once a password is written down, it is no longer something to be cracked or guessed, but something to be located. A password, which is written down, can be easily found by search through user’s personal stuff, like notebook, desk, diary, or user’s manual [Zviran and Haga 1999]. But in an environment where users are managing multiple passwords for multiple different accounts, they start to use different strategies for coping with the password use and remembrance (usually they write their passwords down or share them with their friends or co-workers) [Grawe-

Table VII. Password Memorability

Answer	Phase	desktop comp.	notebook comp.	mobile phone	tablet device	University account	Facebook account	Twitter account	Google account
Yes	First	2 (6.06%)	4 (12.12%)	3 (9.09%)	2 (6.06%)	7 (21.21%)	6 (18.18%)	3 (9.09%)	7 (21.21%)
	Second	3 (12.50%)	2 (8.33%)	1 (4.17%)	(4.17%)	4 (16.67%)	3 (12.50%)	1 (4.17%)	4 (16.67%)
No	First	14 (42.42%)	25 (75.76%)	20 (60.61%)	3 (9.09%)	23 (69.70%)	24 (72.73%)	9 (27.27%)	20 (60.61%)
	Second	11 (45.83%)	20 (83.33%)	14 (58.33%)	1 (4.17%)	18 (75%)	19 (79.17%)	7 (29.17%)	15 (62.50%)

meyer and Johnson 2011]. In both phases of the study, we asked our participants questions related to password memorability and password write down: “Very often, computer users find it convenient to write down their password for one of these unfortunate times when they forget it. Did you do this too for your password?” and “Have you ever had difficulty remembering your password?”. The answers to both questions are summarised in Table VI and Table VII, respectively. In both tables, the first column represents the answers to the question (Yes, No), the second column represents the phase of the study (First, Second). The rest of the columns represent the percentage of users that chose the specific answer for the specific account. The results in Table VI show that users rarely write their passwords down. Even 18.18% of the users reported that they wrote down their password for their University account, and less than 10% for the rest of the accounts. From the results in Table VII we can realise that 21.21% of the users have problems with remembering their University account, 18.18% for the Facebook account, and 21.21% for the Google account. Seems like the passwords for the University, Facebook, and Google account are the most hard ones to remember. One possible explanation for this may lay in some of our previous findings where we discovered that majority of users’ passwords for these accounts were more complex in their structure than the passwords for the rest of the accounts. These results look promising, compared to the results reported by the authors Zviran and Haga [1999] who found that even 35.5% of the participants in their study wrote down their passwords.

4. CONCLUSION

The study investigates the effect of password security training on user’s practices regarding password creation, password use and management. We performed a pilot study on university students to explore their password characteristics and habits regarding password creating, using, managing and security. We collected our data by means of an online questionnaire. The survey consisted of two phases. Between the two phases, the students received lectures about the importance of creating strong and secure passwords, how to choose such passwords and how to manage them.

The results show that the passwords for the desktop and notebook account in the first phase most commonly consisted of lowercase letters only (15.15% for the desktop and 21.21% for the notebook account), regardless the fact that most of the users rated their data on their desktop and notebook computer as very important. We observe an improvement in the characteristics of the passwords for the desktop and notebook accounts in the second phase of the study: passwords for these accounts were mostly alphanumeric with mixed case letters and numbers (20.83% for the desktop and 29.17% for the notebook account). The results from the first phase show that many users never change their passwords since their first use, or change them less than once a year. The results of the second phase of the study look promising, since even 25% of the users, for the desktop computer, 33.33% for the notebook computer, 20.83% for the University account, 29.17% for the Facebook account, and 25% for the Google account, stated that they changed their passwords because they realized that the old pass-

word was very weak and could compromise their other passwords and accounts. 18.18% of the users reported that they wrote down their password for their University account, and less than 10% for the rest of the accounts. Users stated that they usually have problems remembering their University account (21.21%), their Facebook account (18.18%), and their Google account (21.21%). The lectures and recommendations had a positive effect regarding users' password characteristics and password length, but not quite positive regarding password change. Despite our efforts to educate the users about the importance of frequent password change, a large percent of users did not change their passwords following the lectures. The overall conclusion of this preliminary study is that users still lack of security knowledge. Lax security behaviour regarding rare password change and password write-down, still exists (the results show some improvement though since the research made by Zviran and Haga [1999]).

Our plans for future work include research about possible differences in the quality of passwords between students from different faculties and different fields of study. Also the differences in quality of passwords between organizations with defined security policies and those without one. A flexible password policies tailored to mitigate the risks users actually face in a combination with password checkers can help users create strong and easy-to-remember passwords. This work will serve as a starting point for our further research in this area where we want to determine whether our university password policies are useful to the students, and whether the students can easily apply them or the policies cause them problems when creating and using passwords.

REFERENCES

- Anne Adams, Martina Angela Sasse, and Peter Lunt. 1997. Making Passwords Secure and Usable. In *Proc. of HCI on People and Computers XII (HCI 97)*. Springer-Verlag, 1–19.
- Matt Bishop and Daniel V Klein. 1995. Improving system security via proactive password checking. *Computers & Security* 14, 3 (1995), 233–249.
- Rachna Dhamija and Adrian Perrig. 2000. Deja Vu: A User Study Using Images for Authentication. In *Proc. of the 9th Conference on USENIX Security Symposium - Volume 9*. USENIX Association, 4–4.
- Serge Egelman, Andreas Sotirakopoulos, Ildar Muslukhov, Konstantin Beznosov, and Cormac Herley. 2013. Does My Password Go Up to Eleven?: The Impact of Password Meters on Password Selection. In *Proc. of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, 2379–2388.
- Beate Grawemeyer and Hilary Johnson. 2011. Using and Managing Multiple Passwords: A Week to a View. *Interact. Comput.* 23, 3 (2011), 256–267.
- Delbert Hart. 2008. Attitudes and Practices of Students Towards Password Security. *J. Comput. Sci. Coll.* 23, 5 (2008), 169–174.
- A.M. Horcher and G P Tejay. 2009. Building a better password: The role of cognitive load in information security training. (2009).
- Markus Jakobsson and Mayank Dhiman. 2013. The Benefits of Understanding Passwords. In *Mobile Authentication SE - 2*. Springer New York, 5–24.
- Robert Morris and Ken Thompson. 1979. Password Security: A Case History. *Commun. ACM* 22, 11 (Nov. 1979), 594–597.
- Gilbert Notoatmodjo and Clark Thomborson. 2009. Passwords and Perceptions. In *Proc. of the Seventh Australasian Conference on Information Security - Volume 98 (AISC '09)*. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 71–78.
- P. Cisar and S. Maravic Cisar. 2007. Password a Form of Authentication. (2007).
- M A Sasse, S Brostoff, and D Weirich. 2001. Transforming the Weakest Link a Human/Computer Interaction Approach to Usable and Effective Security. *BT Technology Journal* 19, 3 (2001), 122–131.
- Viktor Taneski, Boštjan Brumen, and Marjan Heričko. 2014. Password security no change in 35 years?. In *Proc. of The 37th International ICT Convention MIPRO 2014*. IEEE, 1507–1512.
- Moshe Zviran and William J Haga. 1990. Cognitive passwords: The key to easy access control. *Computers & Security* 9, 8 (1990), 723–736.
- Moshe Zviran and William J Haga. 1999. Password Security: An Empirical Study. *J. Manage. Inf. Syst.* 15, 4 (1999), 161–185.
- M. Zviran and W. Haga J. 1993. A Comparison of Password Techniques for Multilevel Authentication Mechanisms. 36, 3 (1993).