

Zoran Budimac, Marjan Heričko (Eds.)

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

SQAMIA 2015

Maribor, Slovenia, June 8th – 10th, 2015

Proceedings

Institute of Informatics, Faculty of Electrical Engineering and Computer Science
University of Maribor, Slovenia

2015

Fourth Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications (SQAMIA 2015)

Maribor, Slovenia, June 8th – 10th, 2015

The number of printed copies: 50

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

Marjan Heričko
University of Maribor
Faculty of Electrical Engineering and Computer Science, Institute of Informatics
Smetanova ulica 17, 2000 Maribor, Slovenia
E-mail: marjan.hericko@uni-mb.si

Papers are copyrighted © 2015 by the authors. Copying permitted only for private and academic purposes. This volume is published and copyrighted by its editors. The contents of the published papers express the opinions of their respective authors, not the volume publisher.

CIP - Kataložni zapis o publikaciji
Univerzitetna knjižnica Maribor

659.2:004(082)

WORKSHOP on Software Quality Analysis, Monitoring, Improvement, and Applications (4 ; 2015 ; Maribor) SQAMIA 2015 : proceedings / Fourth Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications, Maribor, Slovenia, June 8th - 10th, 2015 ; Zoran Budimac, Marjan Heričko (eds.). - Maribor : Institute of Informatics , Faculty of Electrical Engineering and Computer Science, 2015

50 izv.

ISBN 978-961-248-485-9
1. Budimac, Zoran
COBISS.SI-ID 82672385

ISBN 978-961-248-485-9



Preface

This volume contains papers presented at the Fourth Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications (SQAMIA 2015). SQAMIA 2015 was held during June 8th – 10th, 2015, at the Institute of Informatics, Faculty of Electrical Engineering and Computer Science, University of Maribor, Maribor, Slovenia.

SQAMIA 2015 continued the tradition of successful SQAMIA workshops previously held in Novi Sad, Serbia (in 2012 and 2013), and in Lovran, Croatia (in 2014). The first SQAMIA workshop was organized within the 5th Balkan Conference in Informatics (BCI 2012). In 2013, SQAMIA became a standalone event intended to be an annual gathering of researchers and practitioners in the field of software quality.

The main objective of the SQAMIA series of workshops is to provide a forum for presentation, discussion and dissemination of the latest scientific achievements in the area of software quality, and to promote and improve interaction and collaboration among scientists and young researchers from the region and beyond. The workshop especially welcomes position papers, papers describing work in progress, tool demonstration papers, technical reports, and papers designed to provoke debate on present knowledge, open questions and future research trends in software quality.

The SQAMIA 2015 workshop consisted of regular sessions with technical contributions reviewed and selected by an international program committee, as well as of six invited talks presented by leading scientists in the research areas of the workshop. In total 10 papers were accepted and published in this proceedings volume. All published papers were double or triple reviewed. We would like to gratefully thank all PC members for submitting careful and timely opinions on the papers.

Our special thanks are also addressed to the program co-chairs, Tihana Galinac Grbac (Croatia), 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 2015 Organizing Committee from the Institute of Informatics, Faculty of Electrical Engineering and Computer Science, University of Maribor, and the Department of Mathematics and Informatics, Faculty of Sciences, University of Novi Sad, especially to its chair Viktor Taneski for his hard work and dedication to make this workshop the best it can be.

Finally, we thank our sponsors for supporting the organization of this event. We are especially thankful for financial support by the project co-founded by the European Social Fund and the Ministry of Education, Science and Sport of the Republic of Slovenia. The project will be implemented within the framework of the Operational Programme for Human Resources Development 2007-2013, 3rd priority axis: Development of human resources and lifelong learning, activity 3.3.: Quality, competitiveness and responsiveness of higher education. The workshop is also endorsed by the EU COST Action IC1202: Timing Analysis on Code-Level (TACLe).

And last, but not least, we thank all the participants of SQAMIA 2015 for their contributions that made all the work that went into SQAMIA 2015 worthwhile.

June 2015

Zoran Budimac
Marjan Heričko

Workshop Organization

General Chair

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

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*)

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

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

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

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

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

Organizing Committee

Viktor Taneski, Chair (*Univ. of Maribor, Slovenia*)

Saša Kuhar (*Univ. of Maribor, Slovenia*)

Katja Kous (*Univ. of Maribor, Slovenia*)

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

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

Organizing Institution

Institute of Informatics, Faculty of Electrical Engineering and Computer Science, University of Maribor, Slovenia

Sponsoring Institutions of SQAMIA 2015

SQAMIA 2015 was partially financially supported by:

Republic of Slovenia, Ministry of Education, Science and Sport

European Social Fund

Table of Contents

○ Compile-time Unit Testing	1
<i>Áron Baráth, Zoltán Porkoláb</i>	
○ Small Scale Analysis of Source Code Quality with regard to Native Android Mobile Applications	9
<i>Martin Gjoshevski, Tina Schweighofer</i>	
○ Data Driven Ecosystem – Perspectives and Problems	17
<i>Hannu Jaakkola, Jaak Henno, Jari Soini</i>	
○ Scientific Software Testing: A Practical Example	27
<i>Bojana Koteska, Ljupco Pejov, Anastas Mishev</i>	
○ Rotation Forest in Software Defect Prediction	35
<i>Goran Mauša, Nikola Bogunović, Tihana Galinac Grbac, Bojana Dalbelo Bašić</i>	
○ Benefits of Using Domain Model Code Generation Framework in Medical Information Systems	45
<i>Petar Rajković, Ivan Petković, Dragan Janković</i>	
○ Towards the Formalization of Software Measurement by Involving Network Theory	53
<i>Gordana Rakić, Zoran Budimac, Miloš Savić, Mirjana Ivanović</i>	
○ Validation of Static Program Analysis Tools by Self-Application: A Case Study	61
<i>Miloš Savić, Mirjana Ivanović</i>	
○ Application Challenges of the I/W/SW-OT Paradigm	69
<i>Muhamed Turkanović, Marko Hölbl</i>	
○ Influence of Cultural Issues on Data Quality Dimensions	75
<i>Tatjana Welzer, Marko Hölbl, Lili Nemeč Zlatolas, Marjan Družovec</i>	

Compile-time Unit Testing

ÁRON BARÁTH and ZOLTÁN PORKOLÁB, Eötvös Loránd University

Unit testing is an essential part in high quality software development. The prerequisite of the system-wide test is that all components are working correctly. Such components are checked by unit tests which are focused on small part of code. Unit tests are isolated from other code fragments and are frequently programmed by using third party tools. In this paper we show how to minimize the number of the required third party tools, and how to automate unit tests. We aimed to express unit tests as part of the source code, and execute them during the compilation to ensure the quality of the code. Two different approaches are presented in this paper: the first is based on the C++11's new mechanisms, such as *constant expressions*, and *static assertions*; the second is based on our experimental programming language, *Welltype*, which offers more flexible unit testing than C++11 – however, both technique are similar. The key in both approaches to get the compiler to execute additional, user defined calculations which can result errors during the compilation. In C++11 the static assertions are used to evaluate parts of the unit tests. Since constant expressions are restricted, in this paper we also present a method how to overcome those restrictions, and how to utilize static assertion with constant expressions. Finally, we describe how our experimental language offers compiler support to evaluate *pure* expressions at compile-time.

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, C++, C++11, Compile-time, Unit testing

1. INTRODUCTION

Modern software development have to take testing into account to ensure the reliability of the software product. Test-driven development [Beck 2003] requires to specificate the new features first by writing new test cases, and after implement it to fulfill the test cases. This method provides clear specification for the new features. Moreover, any reported bugs become test cases, and the way of fixing it is the same. Finally, all written test cases are part of the regression test.

Two major kind of tests are known: black-box and white-box testing [Schach 2002]. The *black-box* tests are focused on the input and the output: for specific input, the specific output must be provided, and no matters how. We could say black-box tests are testing the preconditions and the postconditions [Plosch and Pichler 1999]. The *white-box* tests are dedicated to get as high code coverage as possible by providing different inputs to execute distinct parts of the code. Keeping the white-box tests cases up-to-date requires huge effort when the implementation changes, during a refactor for example.

The unit tests can be handled different ways. The tests can be written by hand as any regular program, and refer to the libraries which are tested. This technique is too unflexible, because numerous additional code is required to make detailed error messages. The more general way is to use an existing *test framework*, because of the detailed error messages, large number of tools which are helpful when

Authors' 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, M. Heričko (eds.): Proceedings of the 4th Workshop of Software Quality, Analysis, Monitoring, Improvement, and Applications (SQAMIA 2015), Maribor, Slovenia, 8.-10.6.2015. Also published online by CEUR Workshop Proceedings (CEUR-WS.org, ISSN 1613-0073)

develop tests. Test frameworks are available for all wide-spread programming language, for example *gtest* [Google 2015] for C++ and *JUnit* [Tahchiev et al. 2010] for Java.

However, the test frameworks are usually third party extensions for a language, sometimes came with a platform-specific library. Furthermore, the created tests are regular programs, which must be executed manually – or they are part of the build system. When the execution of the tests are neglected, the quality of the software questionable. Therefore, we need compiler support or at least some compile-time mechanism to prevent the compilation of the wrong code at all. For example, the *Eiffel* programming language introduced the *Design by Contract* [Meyer 1992].

In this paper we show a new way to write tests cases in C++11, which are checked by the compiler. The method uses features are introduced in C++11: the constant expressions, which are expressions containing only constant expressions or literals, and the compiler is entitled to evaluate them during the compilation; and the static assertions, which are assertions evaluated during the compilation. However, these features are very restricted, and requires preparations in the source code to apply. We present the usage of these mechanisms by an example. Moreover, we present a similar technique in our experimental programming language: our ongoing development in our compiler is to evaluate *pure* expressions during the compilation. With this ability, our compiler will support a rich mechanism for compile-time unit testing.

This paper is organized as follows: In Section 2 we give a detailed guide how to write compile-time unit tests in C++11. The guide is presented with a concrete example. Also, we highlight the limitations of the method. In Section 3 we present our experimental programming language called *Welltype*, and its capability of compile-time unit testing. In Section 4 we describe our future plans to implement additional functionality for the compile-time unit testing. Our paper concludes in Section 5.

2. IMPLEMENTATION IN C++11

C++ is a multi-paradigm programming language designed for high performance programming [Stroustrup 1994; 2013]. In the continuous evolution of the language first object-oriented features have been introduced, later essential elements of generative programming, like templates and the STL library were invented. In recent versions in C++14, functional programming achieved more and more importance [Meyers 2014]. In the same time, C++ compile-time programming proved to be a Turing-complete sublanguage [Veldhuizen 2003], and template metaprogram applications became popular [Alexandrescu 2001].

By design no language support exists in C++ to construct test cases. However, developers can use third-party tools and libraries to write tests. One of these tools is the Google Test, also known as *gtest* [Google 2015; Langr 2013; Sigerud et al. 2013], and the Boost Test Library [Rosental 2007]. All third-party tools carry the same problems:

- it is not ensured the availability on all architectures and operating systems;
- the developer’s responsibility to execute the tests during the development.

The key idea is to relocate the functionality from third-party tools into the C++ compiler. This endeavor can be observed in other C++ specific areas, for example generating domain-specific languages at compile-time [Porkoláb and Sinkovics 2011]; or validating STL usages [Pataki et al. 2010].

The C++11 introduced the the `static_assert` mechanisms, and the programmers can write compile-time assertions. Furthermore, the `constexpr` is introduced for optimization purposes, since the compiler is entitled to evaluate all `constexpr` expressions. Note that, before C++11 the compiler could evaluate functions at compile time as well, but that is done by heavy optimization routines. The `constexpr` is more, because the compiler will accept a function as `constexpr`, when the all called functions are

`constexpr`, and the functions itself has no side-effect. This definition excludes the usage of global variables. Consequently, the evaluation of a `constexpr` function depends only on the arguments. If all arguments are constants (literals, or return value of a `constexpr` function), the function will be evaluated at compile-time. Otherwise, it will be evaluated at run-time. The `constexpr` functions can be named as *pure* functions, since the return value depends on the arguments, and the result is the same every time. Furthermore, the C++14 relax the restrictions in the `constexpr`, so more complicated functions can be written [Smith 2013].

Putting the `static_assert` and the `constexpr` together, we can write tests which are evaluated at compile-time. The compile-time tests are the aid for all problems which came from the third-party tools, because all actions are performed by the compiler. So, there is no additional dependencies, and the C++ project is more portable. Using compile-time tests results more reliability, because the source code will not compile if one of the tests fail.

However, the compile-time tests requires specific preparation in the software project. The declaration of the sample class can be seen in Figure 1. Taking advantage of the behavior of the `constexpr`, all functions are annotated with the `constexpr` where it was possible. The constructors are emphasized, because it is mandatory to construct pair at compile-time. The targeted operation is the `operator<` – as it can be seen in the class declaration, it is a friend and a `constexpr` function.

```
class pair
{
    int x, y;

public:
    constexpr pair() : x(), y() { }
    constexpr pair(int x, int y) : x(x), y(y) { }
    constexpr int get_x() const { return x; }

    friend constexpr bool operator<(const pair & lhs, const pair & rhs);

    friend std::ostream & operator<<(std::ostream & os, const pair & rhs);
};
```

Fig. 1. Preparations in the class declaration.

The implementation of the two declared operations in the class can be seen in Figure 2. The `operator<` has the ordinal behavior of a *less-than* operator. Note that, this function can be evaluated in compile-time and in run-time as well, thus, it can be used in `static_assert` statements.

```
constexpr bool operator<(const pair & lhs, const pair & rhs)
{
    return lhs.x < rhs.x || (lhs.x == rhs.x && lhs.y < rhs.y);
}

std::ostream & operator<<(std::ostream & os, const pair & rhs)
{
    return os << '(' << rhs.x << ", " << rhs.y << ')';
}
```

Fig. 2. Trivial implementation of the operators which belong to the pair class.

In the sample code above, the implementation of the operator< is correct, however, the correctness of the operator is not trivial in most of the cases. Since the sample less-than operator can be evaluated at compile-time, and pair objects can be constructed at compile-time (see the constexpr), the code can be seen in Figure 3 is valid. Furthermore, the compiler evaluates them, and checks the correctness of the less-than operator based on the samples.

```
static_assert(!(pair() < pair()), "OK");
static_assert(pair() < pair(1, 0), "OK");
static_assert(pair(2, 5) < pair(5, 2), "OK");
static_assert(pair(2, 5) < pair(2, 6), "OK");
```

Fig. 3. Static assertions which pass at compile-time.

The counter-assertions, which fail on evaluation, can be seen in Figure 4. These assertions are intentionally wrong, and they are constructed to demonstrate the correctness of the static_assert.

```
//static_assert(pair() < pair(), "fail");
//static_assert(pair(3, 0) < pair(), "fail");
```

Fig. 4. Static assertions which *fail* at compile-time.

Note that, the static assertions in Figure 3 and in Figure 4 are placed right after the implementation of the less-than operator. Unlike the regular assert, the static_assert can be placed everywhere, so the static assertions are placed outside all functions.

Using this technique, any test cases can be written inside the source code, and the compiler will evaluate them during the compilation. Furthermore, using the constexpr is a good practice, since the *pure* functions can be validated. Thus, the validation of the pure functions is an additional test, and also it is a motivation to write critical functions – such as less-than operator – side-effect free.

```
constexpr bool test_some(int from, int to)
{
    return (from>to) || (pair(from, 0)<pair(from+1, 0) && test_some(from+1, to));
}

static_assert(test_some(-100, 100), "test some");
```

Fig. 5. Generating test cases with recursive constexpr function.

Furthermore, test cases can be generated during compilation with constexpr functions. An example for the test case generator function can be seen in Figure 5. The sample generates 200 different test cases in total. However, different methods can be designed to increase the coverage.

2.1 Limitations

Unfortunately, the constexpr itself is a limitation, since only constexpr functions can be tested at compile-time. Many other functions, which use input and output (for example: file I/O, lock mechanisms, networking), and implement intentionally impure functionality (for example: random number generator, retrieving current time) can not be tested.

An other limitation is the standard library, because the STL containers can not be used in static assertions due to the lack of constexpr. But user-defined containers are permitted to use.

The assertions themselves can be wrong. However, this could be true for any other tests, so this is not a real limitation.

3. IMPLEMENTATION IN WELLTYPE

Our experimental programming language called Welltype [Baráth 2015], which is an imperative programming language with strict syntax and semantics. The language provides a strong static type system, in order to keep the source code clear and easily understandable. Due to the restrictions in the semantics, the language is resistant to numerous syntactical and semantical vulnerabilities [Baráth and Porkoláb 2014].

The Welltype programming language handles the difference between the *pure* and the *impure* functions. This information is explicitly written in the source code – a function without any from these two attributes defaulted to pure to prevent misuse. The pure functions in Welltype are nearly the same as the `constexpr` functions in C++14. The restrictions for the pure functions are clear: impure functions can not be used, usage of the global variables are forbidden. The compiler uses this information in several checks: for example, only pure functions can be used in *assertions*, and in *pre-* and *post conditions*. This is consequent, because it is unacceptable when the expression in an assertion has side-effects. The meaning of the whole program can be totally different with and without assertions.

We are implementing a new feature in our programming language to support compile-time testing. The initial version uses the pure function mechanism, because the returned value of the pure functions are only depends on the arguments. Furthermore, the returned value is always the same, when the function get the same arguments. Due to this property, the testing of the pure functions are established.

The default output of our compiler is a binary which can be executed on our virtual machine. This virtual machine has some special instructions which needed to execute the compiled Welltype programs efficiently. Since the compiled code can be executed on a virtual machine, and this is part of the compiler, the compiler itself can execute (almost) any programs during the compilation – even if the compiler is in cross-compiler mode.

Putting the parts together, we can implement a compile-time validation mechanism to execute the tests. If one of the tests fail, the output of the compiler will be dropped, due to the failed test cases. The pure functions are important components in the mechanism, because the test results can be trusted for pure functions only.

A synthetic sample code can be seen in Figure 6.

```
pure function div(int n, int m) : int
pre { m != 0 }
post(R) { n, m ; old n >= R*old m &&& old n < (R+1)*old m }
assert { div(10, 2)==5 }
assert { div(0, 1)==0 }
assert { div(16, 6)==2 }
assert { div(16, 5)==3 }
assert { div(16, 4)==4 }
//assert { div(1, 0)==0 } // PreconditionViolationException
//assert { div(1, 1)==2 } // AssertionFailedException
assert { div(2, 0)==0 ; PreconditionViolationException }
{
    return n/m;
}
```

Fig. 6. An example to present the elements of the compile-time unit testing in Welltype.

The compiler will gather the `assert` clauses during the compilation, and generates an other executable program in the memory. This new program will contain all of the `assert` clauses for every pure functions. When the new program is generated, the compiler will pass the binary to the virtual machine. If the virtual machine successfully executed the program, it means, all assertions are passed. It is important to notice, only the compiler program is used during the compilation and assertion evaluation. So, the tests will be checked at compile-time. Whenever an assertion fails while running the tests in the virtual machine, the output will be dropped. This is the expected behavior, because the implementation contains one or more errors.

The internal program, which checks the assertion clauses, will be built with the following algorithm:

- (1) gather all pure functions with at least one `assert` clause,
- (2) generate a function for each `assert` clause, which contains a regular assertion with the same condition,
- (3) if the `assert` clause contains an expected exception, add a single exception handler with the desired exception,
- (4) call all generated functions in the `main` block.

If the assertion in a generated function fails, an `AssertionFailedException` will be raised. This exception will be passed through the `main`, and will be caught by the virtual machine. This error can be easily processed in the caller.

3.1 Limitations

The solution for the compile-time testing in Welltype uses the pure function mechanism – since the result can be guaranteed only for the pure functions – the same problem raised as in C++. In the current situation this limitation can be removed only at the expense of reliability. However, the reliability is the essence of this method.

Another limitation is the usage of external functions and types. Since any concrete implementation can be loaded into the virtual machine, the executed test will be meaningless despite of they necessarily be pure functions. Also, the dependencies will be required to compile a program, which is not acceptable. This limitation can be solved, if the *pre-* and *post-conditions* could be stored in the binary, and these conditions will be validated by the dynamic program loader, and compiler could generate mock objects for the conditions. Thus the testing mechanism could be more flexible.

4. FUTURE WORK

Our current method requires handmade test cases, even in case of the recursive test case in Figure 5. Using template metaprogramming, we can improve the method with automatic test case generation. For example, the input for the `operator<` can be generated by random numbers, and the developer have to write only a logical formula, such as `!(a<b) || pair(a, 0)<pair(b, 0)` – or a similar formula in a general case. Furthermore, we are intended to apply axiom-based test generation [Bagge et al. 2011]. Thus the test cases could be more automated.

Our future plans to improve the Welltype compiler is to make able the compiler to generate white-box test cases which satisfy the pre- and the postconditions. Since the precondition defines an acceptable set of input, and the postcondition is a formal specification for the output, it is possible to generate test cases using the symbolic execution [King 1976]. This could be an outstanding and unique feature in our compiler to support the test-driven development.

5. CONCLUSION

In this paper we introduced a method with the related preparations to able to write compile-time unit tests in C++11. The method based on two new features are introduced in C++11: the constant expressions (`constexpr`) and the static assertions (`static_assert`). The reason of the selection was the fact they are evaluated by the compiler during the compilation. When the static assertion fails, the faulty source code can not be compiled. This feature is a great addition compared to any third party test frameworks. Also, we presented a basic test case generation by example – and it worked as expected. However, more complex generators could be written. The limitations of this method is summarized.

We introduced the abilities in our experimental programming language to describing compile-time tests. This language is called Welltype. The basic concept was to entitle the compiler to evaluate all pure expressions in the source code, since the result only depends on the arguments. Moreover, our language supports to attach assertions to function declarations. These asserts will perform as "static assertions" in C++11, but it uses a more flexible approach. However, the implementation is an ongoing work. The limitations of this method is summarized as well.

Our research is continued by constructing more advanced mechanisms in C++11 (or in C++14) to provide more flexible and richer kit for software developers. We intended to include template metaprograms to generate test cases. Also, in our Welltype compiler we intended to implement a white-box test case generator to reach a very high test coverage automatically.

REFERENCES

- Andrei Alexandrescu. 2001. *Modern C++ Design: Generic Programming and Design Patterns Applied*. Addison-Wesley, Boston, MA.
- Anya Helene Bagge, Valentin David, and Magne Haveraaen. 2011. Testing with Axioms in C++ 2011. *Journal of Object Technology* 10, 10 (2011), 1–32.
- Áron Baráth. 2014-2015. Welltype. (2014-2015). <http://baratharon.web.elte.hu/welltype>.
- Áron Baráth and Zoltán Porkoláb. 2014. Towards Safer Programming Language Constructs. In *10th Joint Conference on Mathematics and Computer Science*. 25.
- Kent Beck. 2003. *Test-driven development: by example*. Addison-Wesley Professional.
- Google. 2015. Google Test. (2015). <https://code.google.com/p/googletest>.
- James C King. 1976. Symbolic execution and program testing. *Commun. ACM* 19, 7 (1976), 385–394.
- Jeff Langr. 2013. *Modern C++ Programming with Test-driven Development: Code Better, Sleep Better*. Pragmatic Bookshelf.
- Bertrand Meyer. 1992. Applying 'Design by Contract'. *Computer* 25, 10 (1992), 40–51.
- Scott Meyers. 2014. *Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14*. O'Reilly Media.
- Norbert Pataki, Zsolt Szűgyi, and Gergő Dévai. 2010. C++ Standard Template Library in a Safer Way. In *Proc. of Workshop on Generative Technologies 2010 (WGT 2010)*. 46–55.
- Reinhold Plosch and Josef Pichler. 1999. Contracts: From analysis to C++ implementation. In *Technology of Object-Oriented Languages and Systems, 1999. TOOLS 30 Proceedings*. IEEE, 248–257.
- Zoltán Porkoláb and Ábel Sinkovics. 2011. Domain-specific language integration with compile-time parser generator library. *ACM SIGPLAN Notices* 46, 2 (2011), 137–146.
- Gennadiy Rosental. 2001-2007. The Boost Test Library. (2001-2007). http://www.boost.org/doc/libs/1_57_0/libs/test/doc/html/utf.html.
- Stephen R Schach. 2002. *Object-oriented and classical software engineering*. Vol. 6. McGraw-Hill New York.
- Katarina Sigerud, Wojciech Sliwinski, J Nguyen Xuan, S Deghaye, X Piroux, V Baggiolini, JC Bau, Gennady Sivatskiy, and Ilia Yastrebov. 2013. *Tools and rules to encourage quality for C/C++ software*. Technical Report.
- Richard Smith. 2013. Relaxing the constraint on constexpr functions. (2013). <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3652.html>.
- Bjarne Stroustrup. 1994. *Design and Evolution of C++*. Addison-Wesley.
- Bjarne Stroustrup. 2013. *The C++ Programming Language, 4th Edition*. Addison-Wesley Professional.
- Petar Tahchiev, Felipe Leme, Vincent Massol, and Gary Gregory. 2010. *JUnit in action*. Manning Publications Co.
- Todd L. Veldhuizen. 2003. *C++ Templates are Turing Complete*. Technical Report.

Small Scale Analysis of Source Code Quality with regard to Native Android Mobile Applications

MARTIN GJOSHEVSKI AND TINA SCHWEIGHOFER, University of Maribor

The popularity of smart phones and mobile applications is growing every day. Every day, new or updated mobile applications are being submitted to different mobile stores. The rapidly increasing number of mobile applications has led to an increase in the interest in overall source code quality. Therefore, we will present a case study, where we analyzed different open source Android mobile applications from different domains and different sizes. They were analyzed using the SonarQube platform, based on the SQALE method. We were aiming to research the overall code quality, the connection between lines of code and technical depth and the most common issues facing mobile applications. The results show that the majority of applications tend to have similar code issues and potential difficulties when it comes to maintenance or updates.

General Terms: Mobile application testing

Additional Key Words and Phrases: source code quality, analysis, technical depth, SQALE

1. INTRODUCTION

In light of the popularity that smart phones and mobile applications have achieved, and intrigued by the constant increase in the number of mobile applications submitted to mobile stores (Statista, 2015), we raised the question of whether this trend has an impact on the overall quality of source code. We were concerned that many mobile application developers may have been focused mostly on fast builds and releases and neglected testing and good design practices, and that this could lead to further difficulties down the road with regard to maintenance and the long-term success of the applications.

Two separate studies in the field of source quality in mobile applications (Syer, Nagappan, Adams, & Hassan, 2014) and (Pocatilu, 2006) have verified that using classical object oriented metrics and approaches for measuring the code quality of desktop and web applications can be used to measure the source code quality of mobile phone applications. A study performed at the University of Maribor (Jošt, Huber, & Hericko, 2013), dealt with a similar issue: two hypothesis were tested, of which the first -- which claimed that using object oriented metrics for the analysis of mobile application code quality does not deter from their usage for the source code quality of desktop applications -- was confirmed. However, the second hypothesis, which claimed that results from analyzing source code quality of equivalent mobile applications developed for different platforms would be undifferentiated, was rejected. The previously mentioned study (Syer, Nagappan, Adams, & Hassan, 2014), has also proven that classical relations between metrics, like “*high coupling low cohesion*,” which are eligible for the source code quality of desktop and web applications and are eligible for mobile applications as well. Their study also found that the claim that “*more code, less quality*” was true in 3 out of 5 applications.

2. BACKGROUND

Static code analysis is a technique for the evaluation of code quality that is performed by analyzing the source code or the binary code, with no need of actually running the code (Michael & Laurie, 2007). The advantage of using tools for static code analysis has been recognized by numerous companies and according to the VDC Research Group (Girad & Rommel, 2013), the market share is expected to grow, on

Authors' addresses: M. Gjoshevski, Faculty of Electrical Engineering and Computer Science, University of Maribor, Smetanova 17, 2000 Maribor, Slovenia; email: martin.gjoshevski@student.um.si; T. Schweighofer, Faculty of Electrical Engineering and Computer Science, University of Maribor, Smetanova 17, 2000 Maribor, Slovenia; email: tina.schweighofer@um.si.

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

In: Z. Budimac, M. Heričko (eds.): Proceedings of the 4th Workshop of Software Quality, Analysis, Monitoring, Improvement, and Applications (SQAMIA 2015), Maribor, Slovenia, 8.-10.6.2015. Also published online by CEUR Workshop Proceedings (CEUR-WS.org, ISSN 1613-0073)

average, by 15% annually. This amount of growth and concomitant demand has led to well-developed tools for static code analysis that have become powerful and feasible.


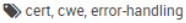
One of the key aspects that defines usable and good tools for static code analysis is that the representation of the results has to be easy to read and understand by the developers (Gomes, Morgado, Gomes, & Moreira, 2015). One way of representing the result is by using technical debt(TD) which is a metaphor reflecting technical compromises that can yield short-term benefits but may hurt the long-term health of a software system(Li, Avgeriou, & Liang, 2015).

2.1 Standards, methods and tools

In this field, there are some relevant standards and methods that support the evaluation of software code quality. ISO/IEC 9126 and its successor ISO/IEC 25010:2011 directly address the issue of system and software quality. We have used ISO/IEC 9126 as a quality model for our analysis, which was performed with SonarQube, based on the SQALE method (Letouzey J.-L. I., 2012).

Software quality assessment, based on life cycle expectations (SQALE), is a method developed by DNV ITGS France that is platform independent and is applicable to any kind of development methodology. The SQALE method estimates the technical debt, which can be presented with custom units, such as cost in money or the time required to remove all the found issues. One of the main four concepts is the quality model, which is organized in three levels. The first and the second level present characteristics such as testability, maintainability etc. and their related sub-characteristics. The third level defines the non-functional requirements or simply the rules, which are usually dependent on the programming language used. The analysis model defines the time required for fixing an issue determined by a rule (Letouzey J.-L. , 2012). Figure 1 gives an example of a defined rule in the SonarQube platform.

Throwable and Error should not be caught
 squid:S1181

 Blocker  Available Since August 30 2013 SonarQube (Java)

Reliability > Exception handling Constant/issue 20min

Throwable is the superclass of all errors and exceptions in Java.

Error is the superclass of all errors, which are not meant to be caught by applications.

Catching either Throwable or Error will also catch OutOfMemoryError and InternalError, from which an application should not attempt to recover.

Only Exception and its subclasses should be caught.

Noncompliant Code Example

```
try { /* ... */ } catch (Throwable t) { /* ... */ }
try { /* ... */ } catch (Error e) { /* ... */ }
```

Compliant Solution

```
try { /* ... */ } catch (Exception e) { /* ... */ }
try { /* ... */ } catch (RuntimeException e) { /* ... */ }
try { /* ... */ } catch (MyException e) { /* ... */ }
```

See

- [MITRE, CWE-396 - Declaration of Catch for Generic Exception](#)
- [CERT, ERR07-J - Do not throw RuntimeException, Exception, or Throwable](#)

Fig. 1. Rule definition in SonarQube

SonarQube is an open source platform that provides a central place for managing source code quality for single or multiple projects. Its extension mechanism allows for the addition of new uncovered

programming languages, tools and features that in conjunction with the default features and tools, such as visual reporting and across projects, time machine etc. provides a genuinely flexible and powerful tool for source code quality analysis (S.A, 2015).

3. CASE STUDY – ANALYSIS OF SOURCE CODE QUALITY

3.1 Case study

A case study is a suitable type of research methodology in the field of software engineering, because it studies contemporary phenomena in its natural context. Primarily, the case study was used for exploratory purposes, but nowadays it is also used for descriptive purposes. Case studies are by definition conducted in a real world environment, and thus have a high degree of realism (Runeson, Hostl 2008).

A case study usually combines five major process steps. The first is case study design, where objectives are defined and a case study is planned. This phase is followed by preparation for data collection, which includes definitions of procedures and protocols. The third step is collecting the evidence, and this is followed by an analysis of the collected data. The final step is the reporting phase of the case study (Runeson & Höst, 2009).

3.2 Objectives of the analysis

The aim of our study was a comparison of the results obtained using the SQALE method with the help of the SonarQube platform, where we analyzed open source mobile applications. We addressed the following research questions:

- RQ1: Is there a major deviation between the overall qualities of the mobile applications source code?
- RQ2: Do lines of code have a direct influence on the technical depth of the mobile applications?
- RQ3: What are the most common issues that occur in the analyzed mobile applications?

3.3 Selection of mobile applications – preparation phase

Our analysis was based on open source mobile applications, so we looked for applications in the F-Droid repository (F-Droid, 2015). F-Droid is a catalogue of open source applications for the Android platform. We selected 30 applications randomly. With the selection we attempted to cover and select applications that cover a vast range of characteristics. Our selection contained applications that belonged to different domain categories, were developed by different developers and differentiated between the lines of code. The complete list of analyzed applications can be seen in Table 1.

Table 1. Selection of applications

Id	Application name	https://github.com	Commit	LOCs
1	K-9 Mail	/k9mail/k-9	c36d2d7a5e5b27f7cfd6892109f9ceb5e49006df	55,603
2	Book Catalogue	/eleybourn/Book-Catalogue	31706472e49500224fd1ed5c9dd15fd253a82081	39,670
3	ChatSecureAndroid	/guardianproject/ ChatSecureAndroid	2a6b6c06fda94f588ad98c97896d0049dd1c2484	36,178
4	TextSecure	/WhisperSystems/TextSecure/	27b5bf54cc2ddd809eedcbb627dbda30d77b2eae	34,429
5	OpenConnect VPN client	/cernekee/ics-openconnect	c8dac1eae12793af78f4bbdc785ebd44e5c016d	25,731
6	AndBible	/mjdenham/and-bible	841eb835d72e2d3488b80bcef754132fa56d6c00	23,478
7	OwnCloud	/owncloud/android	20cd00b22c86ef0ab03ae0b8cd8eedd63844981f	17,660
8	A Comic Viewer	robotmedia/droid-comic-viewer	e9f98610962288b53379f1ac5f889d222f6463e5	16,195
9	Ushahidi	ushahidi/Ushahidi_Android	8c02fecb7dd5b52ef116520c24d4e509e3676c4e	15,950
10	AFWall	/ukanth/afwall	261a7b5919af4459d38a9a6b229f7e22b500c4de	14,006
11	Geopaparazzi	/geopaparazzi/geopaparazzi	0b0e8b7d4285e06e3d0c4a05d804e3328106b7ae	10,892
12	Dudo	/goldenXcode/dudo	9a52e28dfc48e5eb9fc85433072f51203612856	10,643
13	Transdroid	/erickok/transdroid	fef815720e3f46dccc8bb8692f944ba510052ee1	10,430
14	Car Report	*Bitbucket/frigus02/car-report/	b14724ffbe9f73c0654a06c592fbf7e0189ea87e	9411
15	Prey	prey/prey-android-client	9b27da9fb213871ee70744aa493fc5973dc08924	8,788
16	BART Runner	dougkeen/BartRunnerAndroid	5c1ae735e1a301a7141b6e46a26b5864c934778b	5,918
17	oandbackup	/jensstein/oandbackup	a15cbb1c5d91057f9f056f2a71889dc91cabfbe3	5,901

18	LucidBrowsere	/powerpoint45/Lucid-Browser	6d7f86a4f64358135852811e6d9bd39cc5144cb1	5,035
19	CampFahrplan	/tuxmobil/CampFahrplan	a69af4e1f2f147307ae051e3b55649f5345971c8	4,627
20	Aarrddict	/aarrddict/android	17f8892d35483ea6bcdd8dfafe6ef4d5cfbad566	4,008
21	retroarch	/libretro/RetroArch	7e903c248fc3e561133fa9c715db709c766c5984	3,636
22	swftp	/softlayer/swftp	46eabe8bbb06dcf54d5901595eb1b4286afbe5d4	3,098
23	runningLog	/gjoshevski/runningLog	553e9066c84d5e81eb68e176148ec8ea109836ee	3,058
24	OpenDocument	/pear/OpenDocument	8c50b445541a1b3debbd02153de3b853b7e0de8c	2,891
25	bysykklist	/rogerkk/bysykklist-oslo	f714c8e533c099da8d5f35c35e442e428b516cfd	2,006
26	AsciiCam	/dozingcat/AsciiCam	340349e80b6e7cb7f4bb24f164b2165c29bd6060	1,963
27	Wifikeyboard	/darth10/wifikeyboard	e99ee117a00bc9cfd82cc6593b0dce690bcae27	1,909
28	TuxRider	/drodin/TuxRider	36220e5c2f5404bd453f8cb621a1646dd8cf20a4	1,535
29	Presentation	/feelinglucky/Presentation	ef5cc53210283eebad7f2d0bcbe6f6e014b7be17	1,375
30	Submarine	/niparasc/papanikolis-submarine	06140eb34e69a28094628a7f0ac0ff0b01bf2ed3	556

3.4 Setup of the SonarQube quality profile – preparation phase

Our quality profile, in accordance with the SQALE quality model, was in compliance with all Android Lint rules (Lint, 2015). Android Lint is the official tool for validating the code and configuration of android projects. It is usually part of the IDEs that have capabilities for Android development (Eclipse, IntelliJ). Our rules table consisted of 140 lint rules. In addition to the Android Lint rules, we also included the default SonarQube rules created for Java development, known as *Sonar way* (SonarQube, 2015). We have included this set of rules because all of the applications were native applications, written in Java. The *Sonar Way* quality profile contains a total of 197 rules. In the end, we had a complete set of 237 rules. The rules were categorized in line with the ISO/IEC 9126 standard. Table 2 displays the mode used.

Table 2. Quality model

<i>Portability</i>	Compiler
<i>Maintainability</i>	Understandability
	Readability
<i>Security</i>	API abuse
	Errors
	Input validation and representation
	Security features
<i>Efficiency</i>	Memory use
	Processor use
<i>Changeability</i>	Architecture
	Data
	Logic
<i>Reliability</i>	Architecture
	Data
	Exception handling
	Instruction
	Logic
	Synchronization
<i>Testability</i>	Unit tests coverage
	Unit level

4. ANALYSIS OF COLLECTED DATA

After an analysis, the collected data was graded via the SQALE method. The results are presented in Figure 2. As can be seen, the applications are aligned based on lines of code (LOC). The overall result reveals technical depth and is presented in the number of days that are needed in order to remove all of the technical depth. We can see that the applications with the higher number of LOC have more technical depth, and that applications with less LOC effectively do not have technical depth. But in between, by observation, we cannot find a rule that applications with more lines of code have more technical depth.

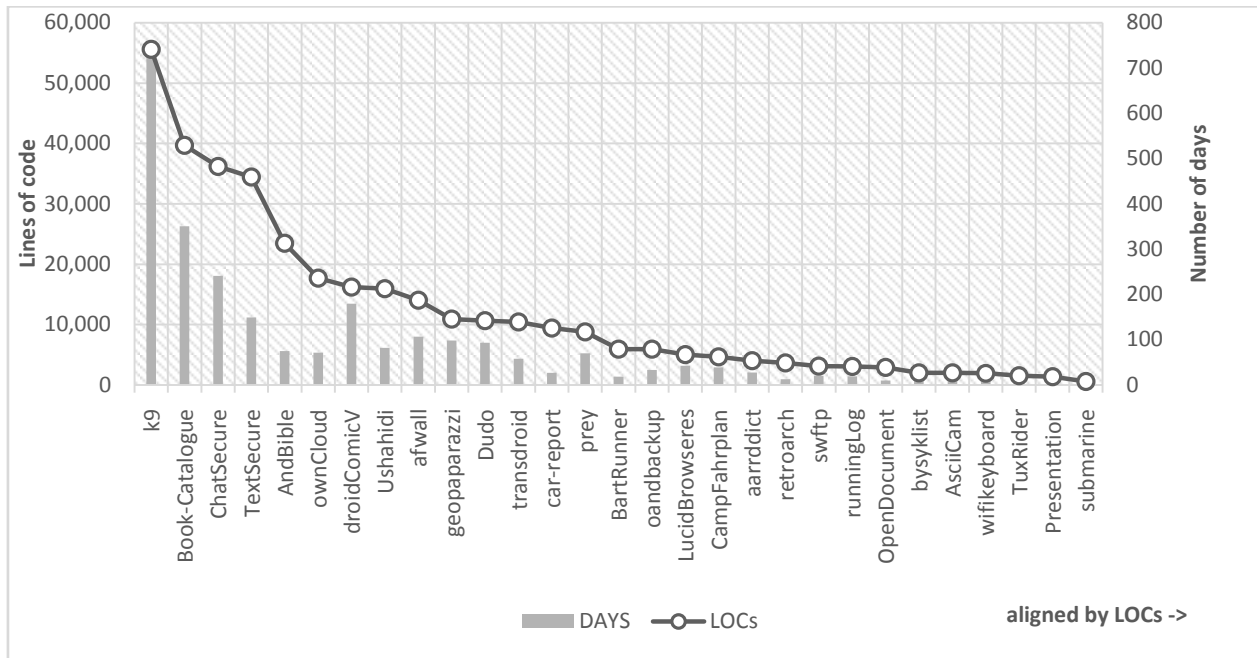


Fig. 2. Initial results

In order to find an answer, supported by data, that would explain the correlation between LOC and technical depth, we presented a new variable. Driven by the assumption that technical depth is correlated with LOC, we calculated an index for each application. The index shows the time required to remove the technical debt for 1,000 lines of code. Our indexes are calculated with the formula presented in Figure 3.

$$i_x = \frac{h_x}{l_x} \times 1000$$

Fig. 3. Normalization formula

Where:

i_x – is the index that represents the time required to eliminate the technical debt for 1000 lines of code.

h_x – is the time required to eliminate the technical debt in the application X.

l_x – is the total number of lines of code (LOC) for the application X.

After applying the formula we get the results presented in Figure 4. The graph shows mobile applications aligned by LOC and time (in days) required to eliminate the technical debt for 1000 lines of code. From the obtained results, based on the data and the applied formula, we were able to see that 13.4% of mobile applications have reached a score that we can categorize as weak and was greater than the upper control limit, which in our case was 9.73 days. On the contrary, 16.6% of the applications received a score that was better than the lower control limit, which was 2.95 days. The average score was 6.79 days, which reveals that the applications had a fair share that can be improved. The upper control limit, lower control limit and mean were calculated based on the results obtained from the analysis. As can we see from the chart, both applications with large LOC and applications with small LOC scored similar scores, so we can conclude, that lines of code do not have a linear correlation with technical depth.

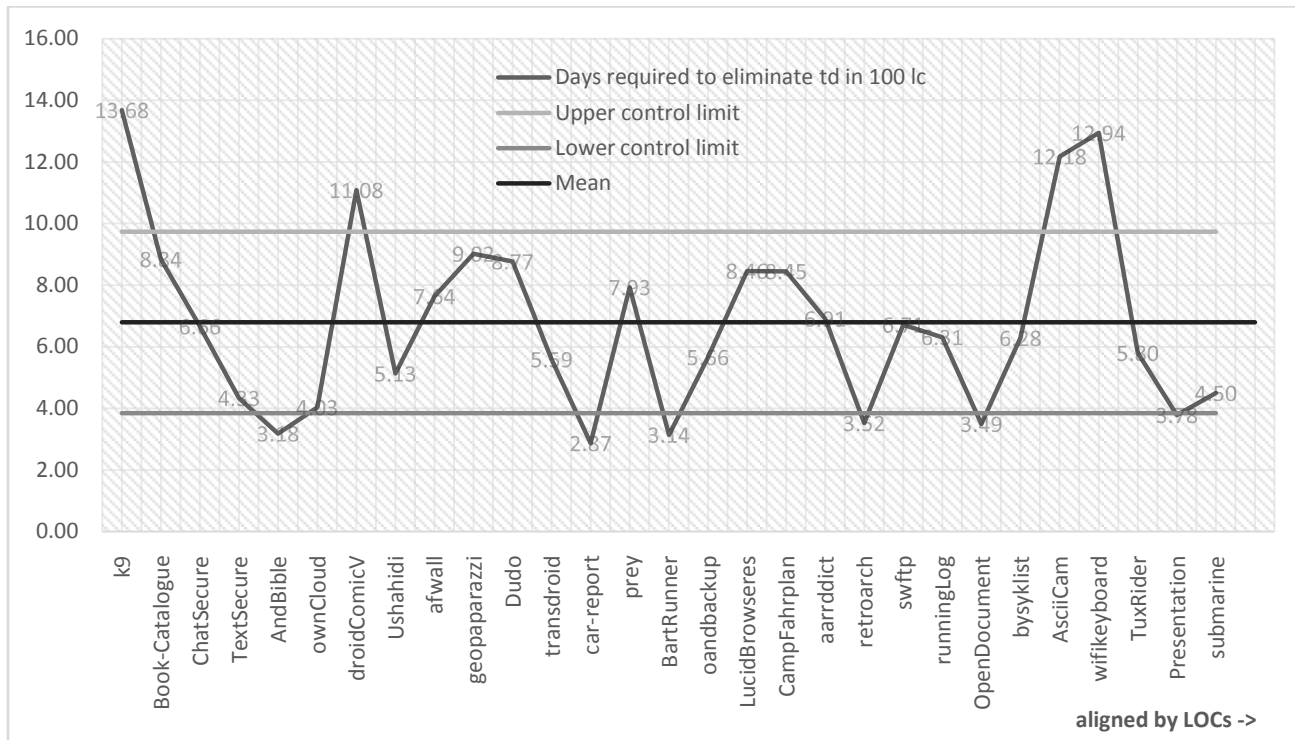


Fig. 4. Control chart

4.1 Problematic code

We also searched for the most common issues that occur in analyzed mobile applications. The detected issues according to our quality profile, based on ISO/IEC 9126, were categorized. Shares were divided among the categories maintainability, changeability, reliability, security, portability, efficiency and testability. The data revealed that the most critical categories were maintainability and changeability, and roughly 91% of all detected issues were categorized in one of these categories. In addition to the mentioned categories, reliability also took a significant share. All shares of the detected issues are presented in the pie chart in Figure 5.

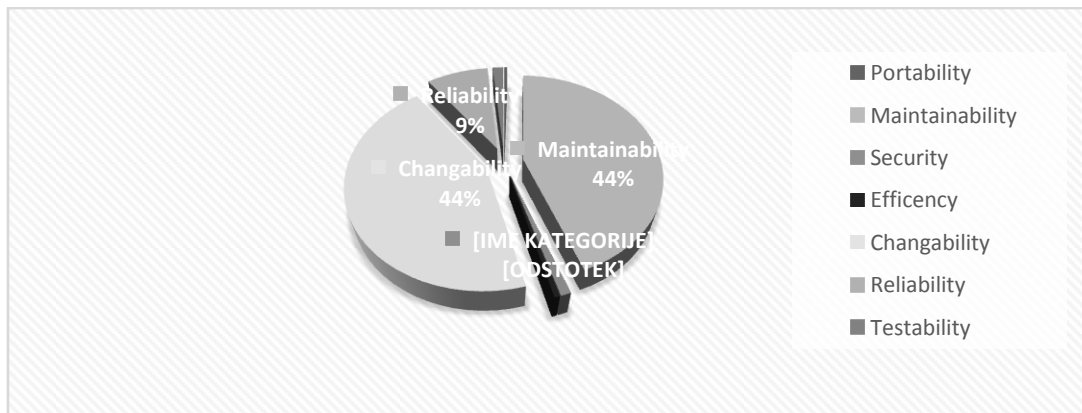


Fig. 5. Code issues by characteristics

5. CASE STUDY RESULTS

RQ1. The first question was aimed at finding if a major deviation occurred between the overall results of the analyzed mobile applications source code. As we can see in Figure 4, the score of 70% of the applications is within the control boundaries and lines of code do not have a direct impact on the overall score. This roughly gives an answer to our question. Another indicator that can support the claim that no major deviation occurs between the overall results are the results from Figure 5, which clearly show that the weakness detected throughout all of the applications belong to the same categories. We believe that this similarity could be due to the fact that most of the developers use IDEs that have code suggestions and validation tools, such as Android Lint, included in their default configuration.

RQ2. The research question was aimed at finding if lines of code in mobile applications have any significant influence on technical depth. We analyzed the applications and presented the technical depth of each mobile application in days. The data was then normalized using the presented formula in Figure 2. We took into consideration that the indexes were within tolerable boundaries and that in most cases the applications had similar code quality. Based on this, the graph in Figure 4 shows that there is no visible correlation between lines of code and technical depth in the analyzed mobile applications.

RQ3. The last question was aimed at finding the most common issues that occurred in analyzed mobile applications. From the 237 rules that we analyzed with our code, we detected only 79. This violations actually represent the 3rd level of the SQALE quality model. In order to get a clear answer to the presented research question, we represented the data with a bar chart. The most commonly detected issue, with a nearly 25% occurrence, was the visibility modifier, followed by other issues such as: avoidance of commented out-lines of code, magic number should not be used, unused private methods and others. All issues, and their respective shares, are presented in Figure 5.

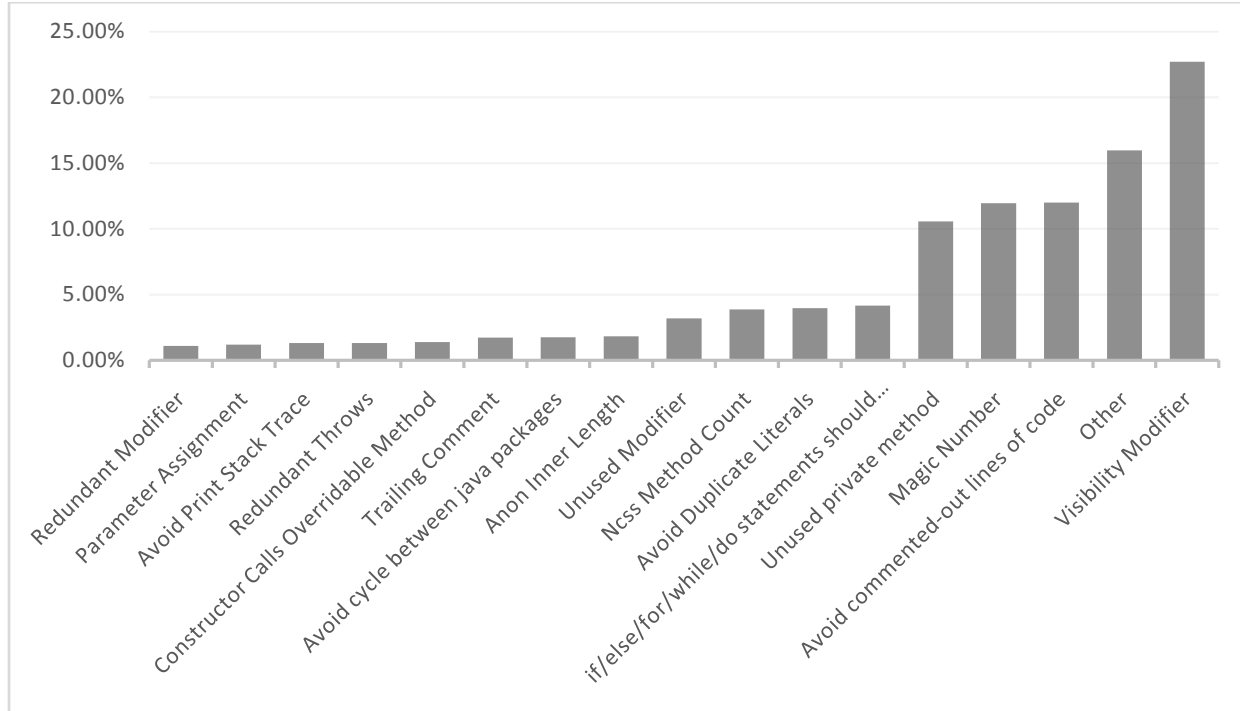


Fig. 5. Most common issues

6. DISCUSSION

Even though we were concerned that code quality would suffer due to the rapid development and race for a share of the growing market, our analysis has given us a results that were not as critical as we expected. In our opinion, there is lot of room for improvement, but the current state of things is bearable and in our subjective opinion is only going to get better, mostly because trends show that the industry is more and more concerned with code quality. This is leading to better quality in the product itself.

This analysis has pointed to a few issues that constantly occur in every project. By dealing with this handful of problems, we could significantly decrease the number of detected issues and can lower the technical debt by *more than 40%*.

We would also like to share our experience of working with the open source platform SonarQube, and would like to stress that it provided a pleasant experience. The platform itself is a powerful tool for managing code quality and in our opinion this tool or similar tools (List of tools for static code analysis, 2015) should become standard practice for managing code quality, which will reflect on the overall quality of the product.

REFERENCES

- F-Droid. (2015, 3 3). Retrieved from Free and Open Source Android App Repository: <https://f-droid.org/>
- Girad, A., & Rommel, C. (2013). The Global Market for Automated Testing and Verification Tools. VDC Group.
- Gomes, I., Morgado, P., Gomes, T., & Moreira, R. (2015, 2 23). An overview on the Static Code Analysis approach in. Retrieved from <http://paginas.fe.up.pt/>: <http://paginas.fe.up.pt/~ei05021/TQSO%20-%20An%20overview%20on%20the%20Static%20Code%20Analysis%20approach%20in%20Software%20Development.pdf>
- Jošt, G., Huber, J., & Hericko, M. (2013). Using Object Oriented Software Metrics for Mobile Application Development. SQAMIA.
- Letouzey, J.-L. (2012, January 27). The SQALE Method - Definition Document - V 1.0.
- Letouzey, J.-L. I. (2012). Managing technical debt with the SQALE method. *IEEE Software*, pp. 44-51.
- Li, Z., Avgeriou, P., & Liang, P. (2015). A systematic mapping study on technical debt and its management. *Journal of Systems and Software*, 193 - 220.
- Lint. (2015, 3 3). Retrieved from Android: <http://tools.android.com/tips/lint>
- List of tools for static code analysis. (2015, January 10). Retrieved from Wikipedia: http://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis
- Michael, G., & Laurie, W. (2007). Toward the Use of Automated Static Analysis Alerts for Early Identification. ICIMP, 18-23.
- Pocatilu, P. (2006). Influencing Factors of Mobile Applications. *ECONOMY INFORMATICS*, 102-104.
- Runeson, P., & Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 131-164.
- S.A, S. (2015, 2 6). Documentation for SonarQube 4.5 and 4.5.x LTS. Retrieved from SonarQube: <http://docs.sonarqube.org/display/SONARQUBE45/Documentation>
- SonarQube. (2015, 3 3). Retrieved from Java Plugin - SonarQube - Confluence: <http://docs.sonarqube.org/display/SONAR/Java+Plugin>
- Statista. (2015, February 25). Retrieved from Number of available apps in the Google Play Store 2015 | Statistic: <http://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/>
- Syer, M. D., Nagappan, M., Adams, B., & Hassan, A. E. (2014). Studying the relationship between source code quality and mobile platform dependence. *Software Quality Journal*, Volum.

Data Driven Ecosystem – Perspectives and Problems

HANNU JAAKKOLA, Tampere University of Technology, Pori Department

JAAK HENNO, Tallinn University of Technology

JARI SOINI, Tampere University of Technology, Pori Department

Our society and business ecosystem is becoming data driven. The value of data is becoming comparable to the value of physical products and becoming an important source of business. Open data itself is seen as a meaningful source of new business, especially for small and medium-sized companies. Open data is purposely aimed at being public. In addition, there is a lot of data used as if it were public – more or less without permission. In addition, the ownership of data has become unclear – the data related to an individual is no longer under the control of the persons themselves. However, declaring data sets to be open and/or allowing access to qualified users does not yet make data useful in practice. On the contrary, this often creates opportunities for misuse and dangers regarding personal security.

Categories and Subject Descriptors: **E [Data]; H.3 [Information Storage and Retrieval]; H.5. [Information interfaces and Presentation]; K.5 [Legal Aspects of Computing]; K.6 [K.6 Management of Computing and Information Systems]; H.1.2 [Models and Principles]; K.6.5 [Security and Protection] *Invasive software, Unauthorized access* - K.7.4 [Professional Ethics]: *Codes of ethics, Codes of good practice*; **K.8 [PERSONAL COMPUTING]: *Games***;**

General Terms: Data, Open Data, Information, Authentication, Invasive software, Unauthorized access, Codes of ethics

1. INTRODUCTION

Our societal and economic ecosystem is becoming data driven at an accelerating speed. In this context, we are not referring to the concept of the “information society” but the importance of data, or to be exact, the cultivated form of it – knowledge based on intelligent data integration – as a key element in the growth of business and welfare of societies. Data possesses business value and changes the traditions of earning models; companies like Facebook and Twitter own huge amounts of user-related profiled and structured data that is valuable in targeted marketing activities or in finding people filling the requirements of a certain kind of profile. In addition, the discussion threads of these services are providing APIs that make the data streams (of public user profiles) more or less open for data analytics; the connection networks (who is connected to whom) are also reasonably easy to analyze as well. To conclude – your value in these (social) networks is your data - not you as a person or contact.

Data is the driving force behind most (future) services and applications. The fast growth of certain innovative businesses is based on data, network infrastructure, and mobility – even in the case of physical products, they are the ultimate source of business. The beneficial use of (social) networks provides a means for communication and availability of potential collaborating (business) partners. In physical products certain properties are built-in and certified – e.g. safety, quality, suitability for use; this is not always true with data. There is also a similarity between data and physical products – e.g. both can be stolen or used in the wrong way or in an illegal / unexpected context. Databases, contact information, or personal profiles are valuable for criminals. Every day we encounter news related to cyber attacks and cyber threats, as well as problems caused by defects in information systems and hijacking of computing resources for illegal or criminal use. New innovations, like IoT, will cause new types of problems: autonomous devices interacting with each other without human control – stories about cyber attacks caused by network-connected devices are already a reality. The global character of cloud services also provides several sources of problems – some related to the safety of data repositories, some to the ownership of the data, and some in processes used to solve disagreements in legal interpretations (e.g. which law is used in globalized implementations). Insurance companies have also noticed new business

Author's address: H. Jaakkola, Tampere University of Technology, Pori Department, email: hannu.jaakkola@tut.fi; J. Henno, Tallinn University of Technology, email: jaak@cc.ttu.ee; Jari Soini, Tampere University of Technology, Pori Department, email: jari.o.soini@tut.fi.

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

In: Z. Budimac, M. Heričko (eds.): Proceedings of the 4th Workshop of Software Quality, Analysis, Monitoring, Improvement, and Applications (SQAMIA 2015), Maribor, Slovenia, 8.-10.6.2015. Also published online by CEUR Workshop Proceedings (CEUR-WS.org, ISSN 1613-0073)

opportunities: different data-related insurances are available that cover loss of data, losses caused by service attacks, etc; in a way, data has become concrete and a physical asset.

Our paper studies the essence of data from different points of view. Section 2 of this paper is focused on data driven changes. We will approach this topic by considering the data driven ecosystem changes driven by hyperscalability. Data-related trends are included in this discussion. Starting with the characteristics of data (Section 3) and continuing with issues related to open data. Section 4 concentrates on problems related to data driven changes. Section 5 concludes the paper.

2. DATA DRIVEN CHANGES

2.1. New ecosystems and hyperscalability

In his blog, Omar Mohaut [2015] has introduced the term “hyperscalable” to point out the importance of data in the growth of business. The only way companies that create physical goods have to scale up their business is increasing productivity (industrialization) and growth of the market. This kind of growth is capital-intensive and the limits are reached quite fast. He lists a set of new-generation companies: Spotify, Skype, Square, PayPal, Facebook, Snapchat, Instagram, Airbnb, Pinterest, Uber, Twitter, Netflix, Kickstarter, Eventbrite, Dropbox, Evernote, BlaBlaCar, Whatsapp, and Booking.com. Their business is based on scalable models and they serve millions of users with very small teams of employees. The growth of their business is not dependent on people in a traditional way, but supported by someone else’s assets as a free lever. Skype has 1,600 employees to operate 40% of international telephone traffic; Airbnb has 600 employees to offer 500,000 rooms for rent without any investment (Hilton, as the biggest hotel chain in the world, has 300,000 employees to operate a hotel business of 680,000 rooms). In Whatsapp, 30 engineers support the message delivery of 7.2 trillion of messages per year (the total amount of traditional SMS messages is 7.5 trillion).

Regarding business based on physical goods, Mohaut compares traditional shopping and e-commerce. E-commerce implements a scalable business model: the website that runs 24*7 all year round and can be reached from all over the world is scalable (data-based). However, physical sales never reach such coverage, because humans as salesmen are not scalable, nor are the buildings needed for stock and shopping centers. A good example of an improved e-commerce model is Alibaba, which handles the data related to goods in the role of broker. As a scalable business concept in real physical goods, Mohaut mentions the concept of franchising: instead of delivering your goods everywhere, you rent out the business concept and brand.

To summarize – what is a hyperscalable business?:

1. A hyperscalable business model is based on intangible assets
2. A hyperscalable business model requires (information) technology as a lever
3. A hyperscalable business model uses the Internet as a free distribution channel

A business is hyperscalable when it “offers value at a near zero cost simultaneously to millions of users with a disproportionately small team.” This business model is not based on the traditional factors of production in economics: land, labor, and capital but on the intelligent and beneficial use of free resources (data, Internet, social networks).

2.2. Trend Analysis as an Evidence

The studies provided by several market analysis companies provide evidence for the progress discussed above. The analysis results also point out the importance of data and (mobile) networking as the driving forces of this progress. SDTimes [2015; 2015a] has analyzed the reports of two leading companies- IDC and Gartner Group. The main trends listed confirm the growing importance of data and the Internet as key factors in progress and changes. The items gathered and combined from these trend lists cover:

1. New technology will take over the market. Growth is focused in 3rd Platform Technologies - mobile devices, cloud services, social technologies, and Big Data.
2. Wireless data growth. Wireless data will balloon to 13% of telecommunications spending. The role of mobile terminals is also changing from speech to data: according to (Finnish) statistics

(Tekniikka & Talous, March 20th, 2015), the average mobile terminal transfer data is 169 MB per day and 5 GB per month; the number of phone calls decreased by 3% in one year (2013-2014) and the number of SMSs by 16%.

3. Cloud services. PaaS, SaaS and IaaS services will remain a hotbed of activity; the highest growth (36%) is projected for IaaS adoption.
4. Big Data and analytics. In addition to the traditional structured and non-structured data, video, audio and image analytics will have growing importance. Data-as-a-Service will forge new Big Data supply chains focused on commercial and open data sets. The IDC (2012) study “Digital Universe” reported fast growth in the amount of data applicable for open analytics (Big Data), which is expected to grow from 130 EB (ExaBytes = 10¹⁸ Bytes) in 2005 to 40,000 EB in 2020.
5. The Internet of Things (IoT). The predictions identified IoT as one of the most important factors for growth of the 3rd Platform. IoT is also based on mobile communication technologies to an increasing extent. According to the current (Finnish) statistics, (Tekniikka & Talous, March 20th, 2015), 9.5% of all mobile devices are used by autonomous collaborating devices (other than mobile terminals). The IDC [2012] study “Digital Universe” predicted/ forecast?? the growth of data produced by autonomous devices: the percentage of such data is expected to grow from 11% to 20% between 2005 and 2020, indicating the breakthrough of the Internet of Things (IoT) technology.
6. Cloud services will become the new data center. Data centers are undergoing a fundamental transformation, with computing and storage capacity moving to cloud, mobile and Big Data-optimized hyperscaled data centers operated by cloud service providers.
7. Security. IDC approaches this important issue with reference to 3rd Platform-optimized security solutions for cloud, mobile and Big Data. It covers mechanisms including biometrics on mobile devices and encryption in the cloud, as well as threat intelligence emerging as an essential Data-as-a-Service category of enterprise-specific threat information. Gartner points out the importance of risk-based security and self-protection. All roads to the digital future lead through security. Because it is impossible to provide a 100% secured environment, there is a need for more sophisticated risk assessment and mitigation tools. Every app will need to be self-aware and self-protecting.
8. Ubiquitous Computing: The growth of importance of mobile devices will continue; organizations have to focus their services and applications on diverse contexts and environments.
9. Advanced, Pervasive and Invisible Analytics: There is a need to manage how best to filter the huge amounts of data coming from the IoT, social media, and wearable devices. Analytics will become deeply but invisibly embedded everywhere.
10. Context-Rich Systems: Applications are able to understand their users and are aware of their surroundings.

The trends point out the importance of mobility, cloud-based solutions and Big Data analytics. An additional factor that has an impact on future life is the easy reachability of the masses – (social) networking and its beneficial use in diverse activities. The 3rd Platform covers millions of apps, billions of users and trillions of autonomous things. Innovative accelerators are robotics, natural interfaces to services, Internet of Things, cognitive systems, and advanced security. Networks, mobile and wireless data transfer and the increasing importance of cloud services are potential sources for increasing security problems – theft of data, stealing of identity, cyber attacks, etc. The use of integrated data (as knowledge) uses diverse sources of data – open and closed. The availability, easy access, quality, and reliability of data will have increasing importance. (OR are growing in importance)

3. DATA CHARACTERISTICS – OPEN DATA AS AN OPPORTUNITY

Open Data is “data that are freely available to everyone to use and republish without restrictions from copyright, patents or other mechanisms of control” [European Union 2014]. This refers to such data sources that are open to the public by the “owner’s” will. “Freely available” is implemented by providing a published interface (API) as access to the data, which is “raw” and needs further processing into useful

form. “Without restrictions” indicates fully free use of the data. In practice, there are restrictions defined by different levels of licensing.

Figure 1 (left side) illustrates the different data categories. In our classification, the term ‘Small Data’ is used to cover all possible data repositories – closed and open. The term ‘Big Data’ illustrates the part of all data that is available for analytics and which provides access for intelligent analysis tools. Openly Available Data covers such data sources that are available in networks and provide a means for monitoring its content. Part of this availability is due to a lack of or insufficient security and part is on purpose. Data sources in this category cover e.g. road cameras, weather stations, technical devices connected to the Internet, www pages, data streams, and the content of social media services. The interpretation of Open Data is explained above. An additional item in this figure is “My Data.” It describes data that is in some way related to a person. By default its ownership is expected to belong to a private person, because it handles his / her private property. However, an increasing amount of this is collected by such information systems that are no longer under the control of an individual: social media services, client and membership cards, location services, etc.

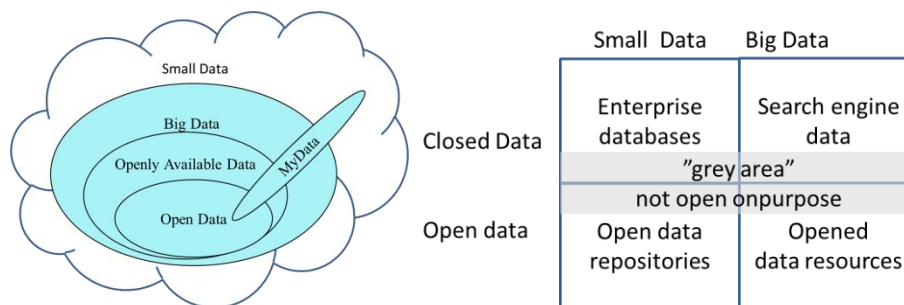


Fig. 1. Data Categories

The left side of Figure 1 approaches the classification from a different point of view. It points out the existence of data that is not open on purpose (by the data owner). This gray area data belongs mainly to the category of openly available data. It would also include such data sources that are available because of insufficient security (by accident) or used against the expectations of the data owner (e.g. www page contents).

The authors of this paper have handled the role of Open Data as a source of future business in their paper [Jaakkola et al. 2014; 2014a]. These findings are summarized below. The starting point is the Digital Agenda of the European Commission [European Union 2014]. The potential of Open Data in the EU area delivers the following benefits:

- Public data has significant potential for re-use in new products and services. Overall economic gains from opening up this resource could amount to € 40 billion a year in EU direct business volume (and € 140 billion indirectly).
- Addressing societal challenges – having more data openly available will help us discover new and innovative solutions;
- Achieving efficiency gains through sharing data inside and between public administrations;
- Fostering participation of citizens in political and social life and increasing transparency of government.

According to our studies, most of the business cases are related to marketing, better understanding of the business environment, availability of economic data related to clients and competitors, and the opportunity to develop context-sensitive services (context is based on the results of open data analysis). Weather and map data were seen as important sources. A lot of potential is also built in MOOCs (Massively Open Online Courses) in education and industrial training.

However, there are also problems. Deloitte Analytics [2014] analyzed 37,500 datasets opened in the UK (data.gov.uk; www.ons.gov.uk; data.london.gov.uk). The study shows the contradiction between the supply and demand of open data. Governmental data are usually collected for official purposes, not for pre-planned business use. The motivation to collect it comes from legal issues. This easily leads to a

situation where the interface (API) to the data becomes complex and the structure of the data is not suitable for effective and beneficiary reuse.

It is also a fact that Open Data is not open without restrictions. The most commonly applied licensing systems in open data are Creative Commons (CC) Licences (<http://www.creativecommons.org>). The other licensing systems, Open Data Commons (ODC) Licences (<http://www.opendatacommons.org/>) and Open Government Licences (<http://www.nationalarchives.gov.uk/doc/open-government-licence/version/2/>) are identical to CC. Common to all of the above-mentioned licences is that they expect the user to acknowledge the source of the information by including an attribution statement specifying the information provider.

4. PROBLEMS

4.1. Malware

Along with the growth of the Internet (currently approx. 40% of the world population already has an Internet connection [Internet Users (2014)]), the misuse of data available on the Internet has also grown. The Independent IT Security Institute AV-TEST registers over 390,000 new malicious programs every day; in 2014, 140000000 new malware items were discovered [Malware Statistics 2015]. The annual damage to the global economy from cyber crime in 2014 is estimated to be 445 USD [Net Losses: Estimating the Global Cost of Cybercrime]. This is already a measurable part of the GDP of many developed countries, e.g. 1.6% of the GDP of Germany, 1.5% of the GDP of the Netherlands and greater than the GDP of many other countries. The actual damage may be substantially higher, since online crime costs are hard to measure - companies, banks, and governments often do not report hacking or the reports are rather ambiguous.

Malware growth is far more rapid than change in any other Internet statistics. According to the McAfee Labs report, 387 new threats appear every minute [McAfee 2015]. And this is only the visible part of the iceberg - the established anti-virus (AV) products are rather weak protection against constant and rapidly increasing threats. According to a recent report from the threat protection company Damballa [Damballa 2014], only 4% of the almost 17,000 weekly malware alerts are investigated. Inside the first hour of submission, AV products missed nearly 70% of malware, only 66% were identified after 24 hours, 72% after a full week, and it took more than six months for AV products to create signatures for 100% of the malicious files used in the study.

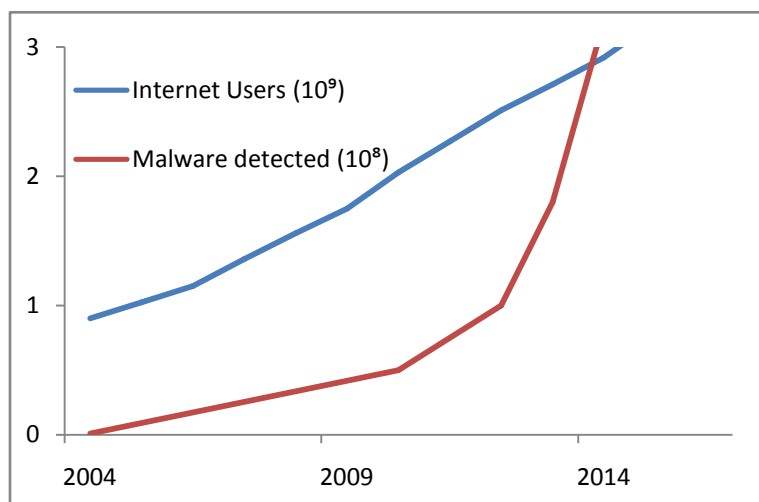


Fig 2. Growth of number of Internet users and detected malware. The exponential growth of new malware guarantees that soon there will be something for every Internet user.

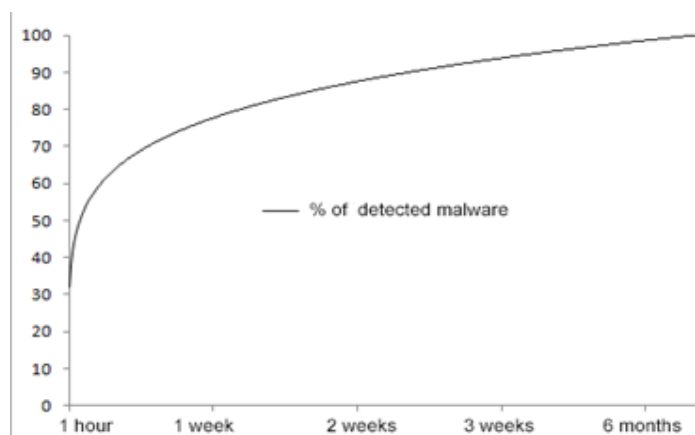


Fig. 3. Percentage of detected malware after hours of discovery. Malware completes the main part of its malicious action in the first thirty minutes after the beginning of the infection [Filiol2005], but probability of its detection in this time is less than 10%.

4.2. Personal Privacy

When you visit some Internet site to get some information, the site also gains some information about you: which web page you came from, which type of browser you are using, and your geographic location. But people often voluntarily expose themselves in some sites much more – who their friends are, what they like, where they have been recently, etc. They assume that this data remains just with this site and often even forget what personal data items they have exposed and where.

However, these items of personal data are valuable for advertisers and spammers and have created a whole new industry of collecting and marketing personal data. As a result, it has become very easy to compile presentations about privacy and security on the Internet. Just declare “there is no such thing anymore.”

There are many examples. For instance, Twitter is currently planning to put trillions of tweets up for sale to data miners [The Guardian, March 2015]; nobody knows what will be revealed from this big bag of data.

Big international companies often consider that they own every piece of data they get and can handle the data just how they like, e.g. expose everything to the United States National Security Agency (NSA) surveillance program PRISM [PRISM], which collects the Internet communications of foreign nationals at several major US companies operating in other countries: Facebook, Apple, Google, Yahoo, Skype, Microsoft. When subsidiaries of these companies are registered in Europe they should also obey European laws. Under EU law, such data export to a third country is legal only if the exporting company can ensure adequate protection for such data, but NSA’s PRISM program and other forms of US surveillance are the exact opposite of adequate protection. In April 2015, Facebook will be challenged by 25000 users in a Viennese Court on violation of European privacy laws [European Court of Justice 2015].

All data will sometimes become obsolete, but even obsolete data can sometimes present a threat to privacy. Facebook ended its e-mail service at the beginning of 2014 [BBC News 2014], thus currently Facebook e-mail addresses are obsolete. However, on March 20, 2015, a list containing 1642 Facebook users’ e-mail addresses was released on an open Internet site. All of them are in standard form `Firstname.Lastname@facebook.com` or use some very similar syntax (`FirstnameLastname@...`), which totally exposes the real name. Unfortunately, this syntax is currently the mandatory standard in many organizations; gone are the days when you could use the address `jaak@...` – far more homely and revealing far less information about your real identity – where the family name is not present.

Taking at random some names from this list and making a Google query returned several websites (Instant PeopleFinders, Whitepages, Spokeoetc) where additional and often very detailed information (even without logging in and paying) were presented, e.g.

Report Includes Available Information on:*****

2 matches for *****

- Current Address:...
- Friends / Family:...
- Phone Numbers:...
- Online Sellers:...
- Email Address:...
- Internet Dates:...
- Marital Status:...
- Old Classmates:...
- Location History:...
- Scammers:...
- Family Members:...
- New Roommates:...

This information was free, but for \$0.85, the site promised to reveal much more.

The first line of the posting containing the list of Facebook addresses was:

“Use for Spam and if you want more msg me...”

Thus this individual knew exactly what s/he was doing - this was just a demonstration of hacking skills in order to get new customers.

Lists of e-mails are published on some sites almost daily, and it is very easy to find more such lists of e-mail addresses – just set up a crawler (using tutorials like [makeuseof 2015]) to search for some properly formatted regular expression, e.g.

```
[ \t:="']+4[0-9]{12}(?:[0-9]{3})?
```

There are sites [e.g. LeakedIn] which provide such lists “just for lulz” (the plural of lol – “laugh out loud” [The Urban Dictionary 2015]).

4.3. New customs

The old generation of digital immigrants [Prensky 2001] watched cinema pictures. How very ‘out’! The new brave generation of digital natives, ‘internauts’, instead play videogames 24/7 [WorldStar 2015] or stream their playing to others to watch on Twitch [Twitch 2015] – the modern incarnation of cinema and TV.

The old generation celebrate birthdays, marriages, births – events which have for us a deep emotional meaning. The new brave generation also have emotional events, but different. For instance, it is very important to increase the number of followers, collect ‘likes’ (who clicked ‘like’ or did something similar on your webpage) and sub/resubs (the analog of likes on Twitch). And if the number of followers/likes/resubs hits some round number, it is cause for celebration. On March 21, the following was pasted on Pastebin:

“Hello everyone and welcome to my 100k Special stream. Hitting 100k Followers is not a small milestone and I wanted to do something crazy for it as thanks for all the support you all have shown me during my time here on Twitch/Youtube.

... I will be streaming for 1 second for every single follower which is 100,000 seconds or 27.7 hours ... every single Sub/Re-sub that happens will add 30 seconds to the total remaining time”

This is followed by the list of games (8) which will be streamed.

4.4. Who has will be given more

The Bible is a wise book – a collection of human experience from many centuries. It emphasizes some principles which are considered important for several occasions:

Matthew 13:12. Whoever has will be given more, and they will have an abundance. Whoever does not have, even what they have will be taken from them.

Matthew 25:29: For whoever has will be given more, and they will have an abundance. Whoever does not have, even what they have will be taken from them. ...

Mark 4:25: For he that has, to him shall be given:

Luke 8:18: Whoever has will be given more; whoever does not have, even what they think they have will be taken from them."

etc.

While the creators/authors of the Bible had to introduce these truths from the historical experience of mankind, nowadays they can be proved.

Consider two computers (black boxes – nothing is known about their structure/functioning), which are connected.

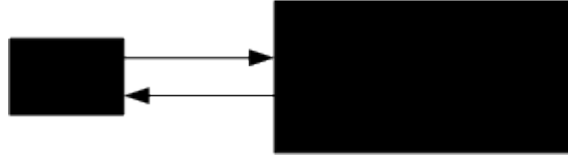


Fig. 4. Connected black boxes; one of them has far more memory.

The boxes exchange messages and store the received responses.

Since they are finite, their number of states is finite and sometimes they go into cycle, start repeating states; repeating states also occurs if some mechanism resets one/both of boxes to some initial state. Obviously, cycling happens first with the box that has less memory. Thus the box with more memory can successfully store all the responses from its little brother and when the little one starts repeating itself, it is concurred? - the big one knows exactly what the responses from the little one will be, so it can use its little brother however it likes. Instead of two black boxes, there is now only one - but with more capabilities, as the big one has the little one as a slave. To quote from Matthew, ‘Whoever has will be given more, and they will have an abundance.’ ‘Whoever does not have, even what they have will be taken from them.’ This situation has been considered by mathematicians in several papers. As long ago as 1973 the following result was proven [Trakhtenbrot & Bardzdin 1973]:

If the size of the input alphabet of an automaton (black box) \mathfrak{M} is m and the size of the output alphabet - n , then for any natural number k one can effectively construct an input word $d(k)$ of length $|d(k)| = 4k^2(\ln nk)m^{2k}$ [which residually distinguishes all automata with k states, i.e. any two automata with k states after getting this input either produce different outputs (they are recognized to be different) or the automata will afterwards act the same way. Since there is only a small number (length is small-power polynomial) of such words, it follows that if automaton \mathfrak{M} is connected to another automaton \mathfrak{M}_1 with more memory and encoded to search the distinguishing word for \mathfrak{M} (with number of states $c_1 * |d(k)| + c_2$, where constant c_1 depends on the encoding of words of length $|d(k)|$, constant c_2 - encoding of search algorithm), automaton \mathfrak{M}_1 can analyze the behavior of automaton \mathfrak{M} , i.e. make \mathfrak{M} do everything that \mathfrak{M}_1 wants. For the case where an upper boundary on the number of states is not known, a polynomial-time probabilistic inference procedure is described in [Rivest & Shappiro 1994].

The coming age of 'Internet of Devices' will be characterized by an increasing number of Internet connected devices with rather little memory - perfect targets which are already being exploited [Goodman 2015]. In 'real' C&C (Conquer and Command) in order to perform an attack, instead of one big computer, a botnet is used - a network of Internet-connected communicating computers. Bots spread themselves from computer to computer searching for vulnerable, unprotected computers to infect and when they find an exposed computer, the machine is infected and then they report back to their master, staying invisible themselves and waiting for further commands. There are always some unprotected computers, e.g. gamers sometimes deliberately disable virus control in order to speed up gameplay. A hacker test in 2012 [Internet Census 2012] found over a million routers that were accessible worldwide. Botnets may have a few hundred or hundreds of thousands of “zombies” infected without their owners' knowledge at their disposal. Botnet creation is 'ridiculously easy' [readwrite 2013, darkreading 2014], but if you do not bother with such activity, you can rent the services of many businesses that operate almost openly or buy an executable program. And if you search, you may find somewhere a code for building a botnet or virus, e.g. code for one of the most sophisticated pieces of malware - Stuxnet - is now freely available in GitHub [Laboratory B 2014].



Fig. 5. Example product from Web store (free to download): flooder - a trojan that allows an attacker to send a massive amount of data to a specific target; user has to specify the victim's IP address, an open port, number of packets and click 'Start'. The web store offers dozens of types of malware - viruses, trojans, scanners, keyloggers, botnets, etc.

4.5. Manage your technological identity

Our identity is determined by our relations with others. Previously, relationships with other members of society were first of all physical. Gradually these physical relations have become replaced with relations based on communication technologies: mail, newspapers, radio/TV; nowadays our most important communication channel/media is the Internet. For many, increasingly their computer/laptop/iPad etc. has become an essential part of their identity – they stare at the screens hoping to get some new like/message/tweet/.. and some cannot even sleep without a mobile in their hand. And yet all the problems with this increasingly important part of our identity are acknowledged painfully. The Internet already influences the psychology of many people, who are constantly checking all their accounts, constantly uploading selfies, constantly sending SMSs. They already live in this virtual world, not in our physical world. When looking at the steady growth of Facebook, Twitter, Pinterest, etc.- the process is escalating. Google has enough memory in its servers to 'pwn' (to conquer to gain ownership [Urban Dictionary 2015a]) every single human memory. Google already understands (somewhat) natural language (queries with full sentences provide better answers than queries containing only keywords) and is rapidly improving its engine. Will we soon get all our truths from Google [ZDNet 2015], and is the whole Internet a Botnet to C&C humanity?

5. CONCLUSIONS

The paper analyzed the essence of a data driven society and ecosystem. The starting point is the phenomenon called hyperscalability. It points out the business value of data and its importance as a source of new business and activities. Open data is seen as a meaningful source of business; in addition, data that is not intended to be open is used for the same purposes. The challenges and threats related to it were discussed.

REFERENCES

- BBC News (2014). Facebook quietly ends email address system. <http://www.bbc.com/news/technology-26332191>. Retrieved March 30th, 2015.
- Chen V. H. H.&Wu Y. (2013).Group identification as a mediator of the effect of players' anonymity on cheating in online games. *Behaviour & Information Technology*, DOI: 10.1080/0144929X.2013.843721.
- Damballa (2014) State of Infections Report Q4 2014. <https://www.damballa.com/state-infections-report-q4-2014>. Retrieved March 30th, 2015.
- darkreading (2014). Researchers Create Legal Botnet Abusing Free Cloud Service Offers. <http://www.darkreading.com/researchers->

- create-legal-botnet-abusing-free-cloud-service-offers/d/d-id/1141418? Retrieved March 30th, 2015.
- Deloitte Analytics (2014). Open Growth – Stimulating Demand for Open Data in the UK. A Briefing Note from Deloitte Analytics. Deloitte LLP, UK.
<http://www.deloitte.com/assets/Dcom-UnitedKingdom/Local%20Assets/Documents/Market%20insights/Deloitte%20Analytics/uk-da-open-growth.pdf> (retrieved March 31st, 2014).
- European Court of Justice hears NSA/PRISM case. http://www.europe-v-facebook.org/PR_CJEU_en.pdf. Retrieved March 30th, 2015.
- European Union (2014). Digital Agenda for Europe: A Europe 2020 Initiative. Open Data. <http://ec.europa.eu/digital-agenda/en/open-data-0> (retrieved March 30th, 2015).
- E. Filiol (2005). Strong Cryptography Armoured Computer Viruses Forbidding Code Analysis: the Bradley Virus. In Turner, Paul & Broucek, Vlasti (eds.), EICAR Best Paper Proceedings, CD-ISBN87-987271-7-6, pp.216-227.
- Marc Goodman (2015). Future Crimes: A journey to the dark side of technology – and how to survive it. Random House, 464 p.
- H. Jaakkola, T. Mäkinen, J. Henno and J. Mäkelä (2014). Open^n. Proceedings of the MIPRO 2014 Conference. Biljanović, P. Mipro and IEEE. ISBN 978-953-233-078-6; pp. 726-733.
- H. Jaakkola, T. Mäkinen and A. Eteläaho (2014a), Open Data – Opportunities and Challenges. In Proceedings of the 15th International Conference on Computer Systems and Technologies - CompSysTech 2014 (Ruse, Bulgaria, June 27, 2014, 2014).ACM.
- IDC (2012). The Digital Universe in 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East. <http://www.emc.com/collateral/analyst-reports/idc-the-digital-universe-in-2020.pdf> (retrieved June 12th, 2014).
- Internet Census 2012. <http://internetcensus2012.bitbucket.org/paper.html>. Retrieved March 30th, 2015.
- Internet Users (2014).<http://www.internetlivestats.com/internet-users/>. Retrieved March 30th, 2015.
- Laboratory B (2014). Stuxnet Source Code on GitHub. <http://www.laboratoryb.org/stuxnet-source-code-on-github/>. Retrieved March 30th, 2015.
- McAfee Labs (Feb 2015). Threats report. <http://www.mcafee.com/mx/resources/misc/infographic-threats-report-q4-2014.pdf>. Retrieved March 30th, 2015.
- makeuseof (2015). How To Build A Basic Web Crawler To Pull Information From A Website. <http://www.makeuseof.com/tag/build-basic-web-crawler-pull-information-website/>
- Malware Statistics (2015). <http://www.av-test.org/en/statistics/>. Retrieved March 30th, 2015.
- Net Losses: Estimating the Global Cost of Cybercrime. Economic impact of cybercrime II. Center for Strategic and International Studies, June 2014, <http://www.mcafee.com/ca/resources/reports/rp-economic-impact-cybercrime2.pdf>. Retrieved March 30th, 2015.
- Mark Prensky (2001). Digital Natives, Digital Immigrants.
<http://www.marcprensky.com/writing/Prensky%20-%20Digital%20Natives,%20Digital%20Immigrants%20-%20Part1.pdf>. Retrieved March 30th, 2015.
- PRISM (2015). http://en.wikipedia.org/wiki/PRISM_%28surveillance_program. Retrieved March 30th, 2015.
- readwrite (2013). How To Build A Botnet In 15 Minutes. <http://readwrite.com/2013/07/31/how-to-build-a-botnet-in-15-minutes>. Retrieved March 30th, 2015.
- R.L. Rivest, R.E. Schapiro (1994). Diversity-Based Inference of Finite Automata. Journal of the ACM, Vol 41 No 3, May 1994, pp. 555-58.
- SDTimes, (2015), IDC's Top 10 technology predictions for 2015. <http://sdtimes.com/idcs-top-10-technology-predictions-2015/>. Retrieved March 26th, 2015.
- SDTimes, (2015a), Gartner's Top 10 strategic technology trends for 2015. <http://sdtimes.com/gartners-top-10-strategic-technology-trends-2015/>. Retrieved March 26th, 2015.
- Shields, Tyler (2015). "The Future of Mobile Security: Securing the Mobile Moment." Forrester Research, February 17, 2015
- The Guardian (2015). Twitter puts trillions of tweets up for sale to data miners.
<http://www.theguardian.com/technology/2015/mar/18/twitter-puts-trillions-tweets-for-sale-data-miners>. Retrieved March 30th, 2015.
- Twitch (2015). <http://www.twitch.tv/>. Retrieved March 30th, 2015.
- B.A. Trakhtenbrot, Ya.M. Bardzdin', "Finite automata.Behaviour and synthesis."North-Holland (1973).
- Veracode, "Average Large Enterprise Has More than 2,000 Unsafe Mobile Apps Installed on Employee Devices." March 11, 2015.
- The Urban Dictionary (2015).<http://www.urbandictionary.com/define.php?term=lulz>. Retrieved March 30th, 2015.
- The Urban Dictionary (2015a).<http://www.urbandictionary.com/define.php?term=pwn>Retrieved March 30th, 2015.
- WordStar (2015). Three Gamers Take Shifts Playing Video Games 24/7 For 500 Days Straight!
<http://www.worldstarhiphop.com/videos/video.php?v=wshhNH6R7gw53ZH9ikA>. Retrieved March 30th, 2015.
- ZDNet (2015).Would you trust Google to decide what is fact and what is not?<http://www.zdnet.com/article/would-you-trust-google-to-decide-what-is-fact-and-what-is-not/>. Retrieved March 30th, 2015.

Scientific Software Testing: A Practical Example

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

When developing scientific applications, scientists are mostly interested in getting scientific research achievements. Usually, the first phase in scientific applications development process is coding. Testing is rarely performed or it is not performed if the final result is correct. Many bugs are found later and problems arise when no software documentation can be found. The goal of this paper is to improve the testing of scientific applications by describing the full testing process which follows the software engineering testing principles. In order to confirm the usefulness of the process, we develop and test an application for solving 1D and 2D Schrödinger equation by using the Discrete Variable Representation method (DVR). Using this testing process in practice resulted in requirements specification, test specification and design, finding connection between specified requirements and tests, automated testing and executable tests.

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

General Terms: Verification

Additional Key Words and Phrases: Scientific application, Testing process, Software Engineering, Schrödinger equation, Software quality

1. INTRODUCTION

Scientific applications perform numerical simulations of different natural phenomena in the field of computational physics and chemistry, mathematics, informatics, mechanics, bioinformatics, etc. They are primarily developed for the scientific research community and usually they are used for simulating some I/O and data extensive experiments [Segal 2005]. The performance of such simulation experiments requires powerful supercomputers, high performance or Grid computing [Vecchiola et al. 2009]. Formally, a scientific application is a software applications which turns the object into mathematical models by simulating activities from the real world [Ziff Davis Publishing Holdings 1995].

Scientific applications are different from commercial application, especially in the process of testing and evaluation. They are developed for the specific research community and not evaluated by customers. Testing is usually performed by comparing the obtained results with theory or performed physical experiments. Also, scientists can decide for the correctness of the results visually, by comparing images.

The absence of software engineering testing practices and documentation is confirmed by the results obtained in the survey we conducted among participants in the High Performance - South East Europe

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, M. Heričko (eds.): Proceedings of the 4th Workshop of Software Quality, Analysis, Monitoring, Improvement, and Applications (SQAMIA 2015), Maribor, Slovenia, 8.-10.6.2015. Also published online by CEUR Workshop Proceedings (CEUR-WS.org, ISSN 1613-0073)

(HP-SEE) project [Koteska and Mishev 2013]. What we are trying to achieve is to change these practices and to adapt some testing practices of the software engineering which according to the IEEEStd 610.12-1990, is an application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software [The Institute of Electrical and Electronics Engineers 2015a].

The goal of this paper is to describe a practical example of testing a scientific application. It will help scientists to learn to include software engineering practices in order to change the current testing practices in different stages of scientific applications development process. In our previous paper [Koteska et al. 2015], we proposed a framework for a complete scientific software development process which provides a set of rules, recommendations and software engineering practices for all development stages. The framework follows the incremental software development with changes in the process of evaluation, in test-driven design and short reports after each iteration. Each increment of the development process contains the following 8 phases: planning, requirements definition, system design, test cases design, coding, testing, evaluation, writing short report.

In this paper, we give special emphasis on the testing process in different development stages. In addition, we adapt and modify the existing software engineering practices for developing commercial applications to successfully include them in the requirements specifications, test cases design, testing and evaluation phase. The proposed testing process is validated with the testing of an application for solving 1D and 2D Schrödinger equations by using the DVR numerical method. The application is tested by following the suggested recommendations. As a result, we have specified test cases, testing functions and automated testing.

The structure of the paper is organized as follows. Section 2 discusses the related work in the area of scientific application testing. The testing process as a part of the development framework for scientific applications and the testing of the application for solving 1D and 2D Schrödinger equation by using the DVR method is presented in Section 3. Section 4 is devoted to the conclusion and future work.

2. RELATED WORK

There are several papers in which authors find some inconsistencies and give recommendations for improving the testing of the scientific applications, but no paper describes the full testing process. In [Sanders and Kelly 2008], the authors found out that scientists use testing only to show that their theory is correct, not that the software does not work. They claimed that testing the computational engine is important, but they are aware of their disorganized and wrong testing practices.

Cox and Harris [Cox and Harris 1999] elaborated a general methodology for evaluating the accuracy of the results produced by scientific software which has been developed as a part of the National Physical Laboratory research. Their idea is to generate and use reference data sets and corresponding reference results to undertake black-box testing. According to this method, the data sets and results can be generated in a consistency with functional specification of the software. The obtained results for the reference data sets are compared with the reference results.

The efficiency of inclusion of software engineers in the process of scientific applications testing is confirmed in [Kelly et al. 2011]. The authors said that software engineer brings a toolkit of ideas, and the scientist chooses and fashions the tools into some thing that works for a specific situation. The results are: an approach to software assessment that combines inspection with code execution and the suppression of process-driven testing in favor of goal centric approaches.

In [Hook and Kelly 2009], the authors highlight the two main reasons for poor testing of scientific software: the lack of testing oracles and the large number of tests required when following any standard testing technique described in the software engineering literature. They also suggest a small number of well chosen tests that can help in finding a high percentage of code faults.

Baxter et al. [Baxter et al. 2006] proposed testing according to the test plans which should be developed in the design phase. They emphasize the need for unit testing which should ensure that a particular module is working properly. Also, bugs can be identified early if testing is performed iteratively throughout the development process. The authors suggest applying of data standards and quality addressing through performance optimization.

In [Hatton 1997], the authors described two large large experiments: measuring the consistency of several million lines of scientific software written in C and Fortran 77 by static deep-flow analysis across many different industries and application areas and measuring the level of dynamic disagreement between independent implementations of the same algorithms acting on the same input data with the same parameters in just one of these industrial application areas. Their results showed that commercially released C and Fortran software are full of statically detectable fault and irrespective of the existence of any quality system, level of criticality, or application area.

3. TESTING PROCESS OF SCIENTIFIC APPLICATIONS

3.1 Testing Methodology

Scientific software testing is complex mainly because the results can not be evaluated by users, but the they are often compared with results from real experiments or they are based on specific scientific theory. In our previous paper [Koteska et al. 2015], we have explained the steps we used for developing an application for solving 1D and 2D Schrödinger equation and in this paper we are focused on functional testing of the application.

Our testing paradigm is: test small software pieces and test often. In order to achieve this, we perform unit testing of each module, in each increment of the application development process. Integration testing is also performed at the end of an increment. Functional testing is closely related to functional requirement specification in which all application functionalities (also inputs and outputs of a functions) are described.

Well written requirements lead to well specified test cases. Since our application is split in independent modules, there is a functional requirement for every module. Each functional requirement is identified by a specific id, name, version, history of changes and description.

Test cases are defined by using standardized forms such as [The Institute of Electrical and Electronics Engineers 2015b] and also some templates [Assassa 2015]. They are described also by unique identifier, name, goal, preconditions, execution environment, expected and actual results, status (passed/failed) and history of changes. It is useful to specify the boundary values or range of values when numerical calculations are performed. Test cases should be written before the coding phase.

We recommend white box testing and automated testing. Also, it is very important to eliminate the redundant tests. Testing process is completed when all tests pass successfully and source coverage is achieved. Test automation can be achieved by various testing frameworks. Since our application is written in C language, we recommend the following frameworks (Check, CUnit, AceUnit, CuTest, etc.). These frameworks are well documented and very easy to use. They provide methods and structures, the user should insert code assertions. The connection between modules and functionality of the application is checked by using the integration testing at the end of each iteration.

The errors found in each iteration must be corrected before the next iteration starts. Errors with higher priority, such as critical errors, should be corrected first. Evaluation of results is usually performed by scientists in the research group.

3.2 Scientific Problem - Solving 1D and 2D Schrödinger Equation by using Discrete Variable Representation Method

Schrödinger equation is a partial differential equation which is used to describe the system dynamics at atomic and molecular level.

A time independent Schrödinger equation can be defined as:

$$H * \Psi = E * \Psi \quad (1)$$

where Hamiltonian operator is denoted by H , Ψ is the wave function of the quantum system and E is the energy of the state Ψ .

A time dependent Schrödinger equation can be represented as:

$$i * \hbar \frac{\partial \Psi}{\partial t} = H * \Psi \quad (2)$$

where \hbar is the Planck Constant divided by 2π and $\frac{\partial}{\partial t}$ is a partial derivative with respect to time t .

One of the ways for solving Schrödinger equation by using matrix representation is the discrete variable representation which is based on truncated standard orthonormal polynomial bases and the corresponding Gaussian quadratures. The matrix elements of differential operators are calculated exactly and those of the operators which are local in the coordinate representation, such as the potential energy operator, are calculated approximately by using Gaussian quadrature accuracy. The Gaussian quadrature gives the most accurate approximation to an integral for a given number of points and weights and it contributes to the DVR method accuracy and efficiency [Szalay et al. 2003].

Discrete variable representation (DVR) methods are widely used because they can solve efficiently numerical quantum dynamical problems. DVR's provide simplification of the kinetic energy Hamiltonian matrix elements calculation and potential matrix elements which represent the potential of the DVR. In a case of multi-dimensional systems, the Hamiltonian matrix is sparse matrix and the operation of the Hamiltonian on a vector is fast when DVR's product is calculated [Light and Carrington Jr 2000].

The goal of this paper is to test a scientific application for solving 1D and 2D Schrödinger equation which is developed in increments. A description of the testing process in each increment is given in the following subsection.

3.3 Testing Process

At the beginning of each increment we have specified the functional requirements. We have identified 13 functional requirements in the first increment and 4 in the second increment. Our application is decomposed in independent modules and functional requirements are tightly connected to modules.

The following modules were developed in the first increment: addition of two matrices, multiplication of two matrices, multiplication of scalar and matrix, printing of matrix elements, matrix transposition, making a diagonal matrix, division of vector by scalar, multiplication of matrix by scalar, making an identity matrix, reading of a file content, comparing two rows of an input file, sorting content in a file, calculation of eigenvalues and module for calculating: arrays of x and y (only in 2D) points in finite basis representation (FBR), transforming matrices for transforming x and y points from FBR to discrete value representation (DVR).

In the second increment, we have developed 4 modules: interpolation function (Hooke method), module for finding the local minimum near given point (x,y) , module for transforming the input file of points and function values in those points and module for solving the 1D and 2D Schrödinger equation which

outputs the frequencies of various vibrational transitions and vibrational wavefunctions on 1D or 2D grid of points.

Next stage connected to testing is the designing of test cases for each functional requirement. We have used the following template for defining test cases as shown in Tables I and II. Both test cases are written in the first increment before testing.

Table I. An Example of Test Case Definition - Module for Making Diagonal Matrix

Test Case Id	06			
Test Case Name	Diagonal Matrix			
Short Description	A module for making a diagonal square matrix from an array of numbers. If array is denoted by A, and matrix by M, then $M[i][i]=A[i]$, where n is the size of A and M. The elements which not lie on the main diagonal are 0's. The return element is the matrix M.			
Pre - conditions	An array of n double elements must be provided as an input and size n must be a integer number.			
Step	Action	Expected Result	Status (Pass/Fail)	Comments
1	Initialization of a 2D array M of size n x n	Allocated memory for M with size n x n. Elements are of double type	Pass	A dynamic allocation is used
2	Assigning elements to the matrix M	Elements of the array A are assigned to matrix main diagonal, the other elements are 0's. $M[i][i]=A[i]$, $M[i][j]=0$, $i \neq j$	Pass	/

The next phase is the unit and integration testing of the application. The well specified test cases are very useful for the testing process. In order to perform automatic testing, we use a system for writing, editing and running unit tests in C, called CuTest [CuT 2015]. The testing of a system functionality is finished when the statuses for all steps of the test case which describes that functionality are "pass".

An example of testing function is shown in Figure 1. This test function is connected to test case 13 which is described above. The values of elements of arrays ptsx, fbrtx and matrix Tx are tested. We are comparing the actual and expected results of variables.

One way to evaluate the final results is to test the convergence of the results with increasing the density of grid points which is equivalent to increasing the number of basis functions or to compare them to results obtained for existing application written in another programming language.

4. CONCLUSION AND FUTURE WORK

In this paper, a testing process of scientific applications is described. It is shown that the requirement specification and test cases design phases are tightly connected to the testing phase because we specified test case for each functional requirement and there is a test designed for each test case. We provide an example and identify challenges for scientific application development. We have tested the application for solving 1D and 2D Schrödinger equation by using the DVR method. We specified test cases for each application module and write tests by using the CuTest framework. The testing process is automated since all tests are executed continuously. It is a good practice to repeat the testing of a given function if some code changes are performed inside the function. The most important lessons learned by this research are that current scientific software testing practices must be changed and the software engineering practices can be successfully included in the scientific application testing.

Table II. An Example of Test Case Definition - Module for Generating Arrays of x Values and y Values, Arrays of x and y Values in FBR and Transformation Matrices for Transforming x and y values from FBR to DVR (thcheby).

Test Case Id	13			
Test Case Name	Generating Arrays of x Values and y Values, Arrays of x and y Values in FBR and Transformation Matrices for Transforming x and y values from FBR to DVR.			
Short Description	<p>This module has five input arguments: nx (number of x values - integer) , xmin (minimum value of x - double), xmax (maximum value of x - double), ny (number of y values - integer), ymin (minimum value of y - double), ymax (maximum value of y - double). First, the differences between the maximum and minimum x values (deltax) and the maximum and minimum y (deltay) values are calculated. Next, the array of x values (ptsx) and array of y values (ptsy) are generated by calling the function make_array_ptsxy(nx,nx+1,xmin,deltax) for x values and make_array_ptsxy(ny,ny+1,ymin,deltay) for y values. Then, the elements of arrays of x values (fbrtx) and y values (fbrty) in FBR are calculated by calling the function make_array_fbrtxy(deltax,nx) for x values and make_array_fbrtxy(deltay,ny) for y values. At the end the transformation matrices Tx and Ty are created by calling the function make_matrix_Txy(nx,nx+1) for x values and make_matrix_Txy(ny,ny+1) for y values;</p>			
Pre - conditions	The number of x and y values must be known (nx and ny). Also the minimum and maximum values of x and y must be initialized (xmin, xmax, ymin, ymax).			
Step	Action	Expected Result	Status (Pass/Fail)	Comments
1	Calculation of difference between the maximum and minimum value of x.	deltax=xmax-xmin Elements are of double type	Pass	/
2	Calculation of difference between the maximum and minimum value of y.	deltay=ymax-ymin Elements are of double type	Pass	/
3	Generating the elements of the array ptsx.	ptsx[i]=((i+1)*deltax*1.0)/(nx+1) +xmin Elements are of double type	Pass	The results are tested by testing the function make_array_ptsxy (nx,nx+1,xmin,deltax)
4	Generating the elements of the array ptsy.	ptsy[i]=((i+1)*deltay*1.0)/(ny+1) +ymin Elements are of double type	Pass	The results are tested by testing the function make_array_ptsxy (ny,ny+1,ymin,deltay)
5	Generating the elements of the array fbrtx.	fbrtx[i]= (((i+1)*Pi)/deltax) ² Elements are of double type	Pass	The results are tested by testing the function make_array_fbrtxy (deltax,nx)
6	Generating the elements of the array fbrty.	fbrty[i]= (((i+1)*Pi)/deltay) ² Elements are of double type	Pass	The results are tested by testing the function make_array_fbrtxy (deltay,ny)
7	Generating the elements of the matrix Tx.	$Tx[i][j]= \sqrt{2.0/(nx+1)} * \sin((i+1) * (j+1) * \Pi/(nx+1))$ Elements are of double type	Pass	The results are tested by testing the function make_matrix_Txy(nx,nx+1)
8	Generating the elements of the matrix Ty.	$Ty[i][j]= \sqrt{2.0/(ny+1)} * \sin((i+1) * (j+1) * \Pi/(ny+1))$ Elements are of double type	Pass	The results are tested by testing the function make_matrix_Txy(ny,ny+1)

Our further aim is to propose a software development process for scientific applications. Our future work will be directed in empirical research and number of case studies that would be defined with goal to develop software development process and all supporting guidelines, methods and techniques. Also, we will focus on development of more complex and critical scientific applications where the inclusion of software formal engineering practices is indispensable.

```

void Test_thcheby(CuTest *tc)
{
    struct return_objects result=thcheby(10, 1, 5, 10, 2, 6);
    double *actualptsx=result.ptsx;
    double *actualfbrtx=result.fbrtx;
    double **actualTx=result.Tx;
    int i,j;
    double *expectedptsx=malloc(10*sizeof(double));
    double *expectedfbrtx=malloc(10*sizeof(double));
    double **expectedTx=malloc(10*sizeof(double));
    for(i=0; i<10; i++)
    {
        expectedptsx[i]=((i+1)*(5-1)*1.0)/11+1;
        expectedfbrtx[i]=square(((i+1)*M_PI)/(5-1));
        CuAssertTrue( tc, abs(expectedptsx[i]- actualptsx[i])<0.0000001);
        CuAssertTrue( tc, abs(expectedfbrtx[i]- actualfbrtx[i])<0.0000001);
        expectedTx[i] = malloc(10*sizeof(double));

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

```

Fig. 1. Function for Testing the method "thcheby"

REFERENCES

2015. CuTest: C Unit Testing Framework. (March 2015). <http://cutest.sourceforge.net/>
- Dr. Ghazy Assassa. 2015. Software Engineering Test Case Template and Examples. (May 2015). [Online]. Available: <http://faculty.ksu.edu.sa/ghazy/CSC342.Tools/Test%20Case%20Template.pdf>
- Susan M Baxter, Steven W Day, Jacquelyn S Fetrow, and Stephanie J Reisinger. 2006. Scientific software development is not an oxymoron. *PLoS Computational Biology* 2, 9 (2006), e87.
- MG Cox and PM Harris. 1999. Design and use of reference data sets for testing scientific software. *Analytica Chimica Acta* 380, 2 (1999), 339–351.
- Les Hatton. 1997. The T experiments: errors in scientific software. *Computing in Science and Engineering* 4, 2 (1997), 27–38.
- Daniel Hook and Diane Kelly. 2009. Testing for trustworthiness in scientific software. In *Software Engineering for Computational Science and Engineering, 2009. SECSE'09. ICSE Workshop on*. IEEE, 59–64.
- Diane Kelly, Stefan Thorsteinson, and Daniel Hook. 2011. Scientific software testing: analysis with four dimensions. *Software, IEEE* 28, 3 (2011), 84–90.

- 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
- Bojana Koteska, Ljupco Pejov, and Anastas Mishev. 2015. Framework for Developing Scientific Applications- Solving 1D and 2D Schrodinger Equation by using Discrete Variable Representation Method. In *The First International Conference on Advances and Trends in Software Engineering SOFTENG 2015*, in print.
- John C Light and Tucker Carrington Jr. 2000. Discrete-variable representations and their utilization. *Advances in Chemical Physics* 114 (2000), 263–310. <http://dx.doi.org/10.1002/9780470141731.ch4>
- Rebecca Sanders and Diane Kelly. 2008. Dealing with risk in scientific software development. *IEEE software* 25, 4 (2008), 21–28.
- Judith Segal. 2005. When Software Engineers Met Research Scientists: A Case Study. *Empirical Software Engineering* 10, 4 (2005), 517–536. DOI: <http://dx.doi.org/10.1007/s10664-005-3865-y>
- Viktor Szalay, Gábor Czakó, Adám Nagy, Tibor Furtenbacher, and Attila G Császár. 2003. On one-dimensional discrete variable representations with general basis functions. *The Journal of chemical physics* 119, 20 (2003), 10512–10518.
- The Institute of Electrical and Electronics Engineers. 2015a. IEEE Standard Glossary of Software Engineering Terminology. The Institute of Electrical and Electronics Engineers, New York, NY, USA. (March 2015). [Online]. Available: <http://www.idi.ntnu.no/grupper/su/publ/ese/ieee-se-glossary-610.12-1990.pdf>.
- The Institute of Electrical and Electronics Engineers. 2015b. Test Case Specification Template (IEEE 829-1998). The Institute of Electrical and Electronics Engineers, New York, NY, USA. (May 2015). [Online]. Available: <http://www.ufjf.br/eduardo.barrere/files/2011/06/SQETestCaseSpecificationTemplate.pdf>.
- 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.
- Inc. Ziff Davis Publishing Holdings. 1995. PC Magazine. (1995). <http://www.pcmag.com/encyclopedia/term/50872/scientific-application>

Rotation Forest in Software Defect Prediction

GORAN MAUŠA, Faculty of Engineering, University of Rijeka, Rijeka, Croatia

NIKOLA BOGUNOVIĆ, Faculty of Electrical Engineering and Computing, University of Zagreb

TIHANA GALINAC GRBAC, Faculty of Engineering, University of Rijeka

BOJANA DALBELO BAŠIĆ, Faculty of Electrical Engineering and Computing, University of Zagreb

Software Defect Prediction (SDP) deals with localization of potentially faulty areas of the source code. Classification models are the main tool for performing the prediction and the search for a model of utmost performance is an ongoing activity. This paper explores the performance of Rotation Forest classification algorithm in the SDP problem domain. Rotation Forest is a novel algorithm that exhibited excellent performance in several studies. However, it was not systematically used in the SDP. Furthermore, it is very important to perform the case studies in various contexts. This study uses 5 subsequent releases of Eclipse JDT as the objects of the analysis. The performance evaluation is based on comparison with two other, known classification models that exhibited very good performance so far. The results of our case study concur with other studies that recognize the Rotation forest to be the state of the art classification algorithm.

Categories and Subject Descriptors: **D.2.9 [Software Engineering]:** Management—*Software quality assurance (SQA)*; **H.2.8 [Information Systems]:** Database Applications—*Data mining*

Additional Key Words and Phrases: Rotation Forest, Random Forest, Logistic Regression, Software Defect Prediction

1. INTRODUCTION

Software Defect Prediction (SDP) is an evolving research area that aims to improve the software quality assurance activities. It is in search for an effective predictive model that could lead the testing resource allocation towards the software modules that are more likely to contain defects. Empirical studies proved that there is certain regularity in defect distribution. It follows the Pareto principle, meaning that minority of source code (20%) is responsible for majority of defects (80%) [Galinac Grbac et al., 2013]. Many classification models have been used for SDP, with various outcomes. It is important to emphasize that the context of data source may be the cause of inconsistent results in software engineering research [Galinac Grbac and Huljenić, 2014]. Therefore, we need to perform a large number of systematically defined case studies with as much data from various domains in order to achieve generalizability of results.

In this paper, we examine the potential of Rotation Forest (RttFor), a novel classification model. The RttFor achieved some promising results in classification problem domain [Rodríguez et al., 2006]. However, its potential is scarcely examined in the SDP research area. That is why we compare its performances with two known classification models, the Logistic Regression (LogReg) and the Random Forest (RndFor). LogReg is an example of a reliable classification model of good performance in many application domains. RndFor is another novel approach that showed promising results and in many cases outperformed other classification models [Lessmann et al., 2008]. RttFor is similar to RndFor because it uses a number of decision trees to make the prediction. But unlike RndFor, each decision tree uses all the features. Furthermore, it weights each feature using the principal component analysis (PCA) method upon randomly selected groups of input features. That way it maximizes the variance between features and achieves better performance [Amasyali and Ersoy, 2014].

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, 51000 Rijeka, Croatia; email: goran.mausa@riteh.hr; N. Bogunović, Faculty of Electrical Engineering and Computing, Unska 3, 10000 Zagreb, Croatia; email: nikola.bogunovic@fer.hr; T. Galinac Grbac, Faculty of Engineering, Vukovarska 58, 51000 Rijeka, Croatia; email: tihana.galinac@riteh.hr; B. Dalbello Bašić, Faculty of Electrical Engineering and Computing Unska 3, 10000 Zagreb, Croatia; email: bojana.dalbello@fer.hr

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

In: Z. Budimac, M. Heričko (eds.): Proceedings of the 4th Workshop of Software Quality, Analysis, Monitoring, Improvement, and Applications (SQAMIA 2015), Maribor, Slovenia, 8.-10. 6.2015. Also published online by CEUR Workshop Proceedings (CEUR-WS.org, ISSN 1613-0073)

We perform the comparison in the context of five subsequent releases of an open source projects for java development, the Eclipse JDT. The obtained results are evaluated in terms of Accuracy, TPR, FPR, F-measure, Kappa statistics and AUC. The paired T-test indicated RttFor to perform equally good or significantly better than the other two classifiers in all the evaluation metrics, with the exception of TPR. Our findings are consistent with other few case studies that obtained favorable results when comparing RttFor to other classification and even regression models [Pardo et al., 2013].

The structure of this paper is as following: Section 2 provides the promising results achieved by RttFor in classification domain that motivated this study. Section 3 presents the algorithms of the 3 classifiers we compare in more details. The description of our case study is given in section 4. The results are presented and their threats to validity are examined in section 5. Section 6 finally gives the conclusion.

2. BACKGROUND

There are some case studies that achieved promising results when using RttFor algorithm. The authors of the algorithm compared it with Bagging, AdaBoost and RndFor on a random selection of 33 benchmark data sets from UCI machine learning repository [Rodríguez et al., 2006]. These datasets contained from 4 up to 69 features, from 57 up to 20,000 instances and from 2 to 26 distinct classes. Their results indicated that the RttFor outperformed other algorithms, achieving 84 wins and only 2 loses in paired comparison of significant differences between models' accuracy levels. Amasyali and Ersoy [Amasyali and Ersoy, 2014] performed the comparison of Bagging, Random Subspaces, RndFor and RttFor algorithm on 43 datasets from the same UCI repository in terms of accuracy. Each algorithm was used with and without the addition of new, artificially combined features. RttFor with the addition of new features outperformed all the other algorithms and RttFor without the addition of new features was the second best.

There are several studies that used the RttFor for different classification purposes, like image classification [Kuncheva et al., 2010], [Xia et al., 2014], [Zhang, 2013]. There, the RttFor was outperformed by Random Subspace with Support Vector Machine (SVM) and several other algorithms in classification of brain images obtained through functional magnetic resonance imaging [Kuncheva et al., 2010]. The impact of several feature transformation methods was analyzed in the RttFor: PCA that is the default method, maximum noise fraction, independent component analysis and local Fisher discriminant analysis [Xia et al., 2014]. Xia et al. also compared each of these variations of RndFor to CART, Bagging, AdaBoost, SVM and LogReg via Variable Splitting and Augmented Lagrangian (LORSAL) in hyperspectral remote sensing image classification. The default variant of RttFor with PCA outperformed others in terms of accuracy. An interesting cascade classifier ensemble is created by Zhang [Zhang, 2013]. He combined k-Nearest Neighbor (kNN), SVM, Multi-Layer Perceptrons (MLP) and RndFor as the first cascade and RttFor with MLP as the second cascade. Each stage gives a majority vote that is supposed to be above a predefined threshold value. The second cascade is targeting the rejected instances from previous cascade, for which the majority vote was not above that threshold, to further insure the confidence. They achieved a great improvement in reducing the rate of rejected instances and, therefore, minimizing the misclassification cost. All of these studies used the RttFor because its performance was reported to be very good comparing to other classifiers.

To the best of our knowledge, the only use of RttFor in the SDP domain was done by [Palivela et al., 2013]. However, it remained unexplained what is their source of data, what information is stored in it, how many features and how many instances does it contain. They compared several classification algorithms: C4.5, SMO, RttFor, Bagging, AdaBoost M1, RndFor and DBScan, evaluated the performance in terms of 8 different evaluation metrics: Accuracy, True Positive rate, False Positive rate, Recall, F-measure, Kappa statistics and AUC, but lacked the comparison of statistical significance. Nevertheless, their results also indicated the RttFor as the classifier of utmost performance.

3. CLASSIFICATION MODELS

All the algorithms involve building models iteratively upon a training set. The training set contains multiple independent variables and 1 dependent variable that we want to predict. The model is trained on this dataset and then it is evaluated on previously unseen data, i.e. the testing set. In classification domain, the dependent variable is discrete, unlike in regression domain, where it is continuous. In our case, the dependent variable is a binary class, where 1 indicates the presence of a bug and 0 indicates the absence of bugs and the independent variables are source code features. We present the algorithms of all the three classification models in the remainder of this section.

3.1 Logistic Regression

LogReg is a statistical classifier. It is used in various classification problem domains and it is renowned as a robust method. That quality makes this classification algorithm appealing to software engineering domain where data are rarely normally distributed, usually are skewed and contain outliers and missing values. The multivariate LogReg is used for classification problem with multiple independent variables [Tabachnick and Fidell, 2001]. For a training set that consists of a set of features X of size $(N \times n)$, with N being the number of instances and n being the number of features m_k (with $1 \leq k \leq n$), and of dependent variable Y of size n as the binary class vector, the LogReg classification algorithm can be explained in these steps:

- (1) Initiate the search of regression coefficients C_k , where C_0 is the intercept and C_k are the weights for each feature m_k and build the classification model \hat{Y} as:

$$\hat{Y}(m_1, m_2, \dots, m_n) = \frac{e^{C_0 + C_1 m_1 + \dots + C_n m_n}}{1 + e^{C_0 + C_1 m_1 + \dots + C_n m_n}} \quad (1)$$

- (2) Evaluate the model by assessing the natural log likelihood (NLL) between the actual (Y_j) and the predicted (\hat{Y}_j) outcomes for each j -th instance (with $1 \leq j \leq N$) as:

$$NLL = \sum_{j=1}^N [Y_j \ln(\hat{Y}_j) + (1 - Y_j) \ln(1 - \hat{Y}_j)] \quad (2)$$

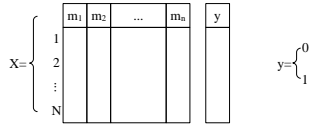
- (3) Optimize the coefficients using maximum likelihood procedure iteratively, until convergence of coefficients is achieved
- (4) The output of classification is the probability that a given instance belongs to the class $\hat{Y}_j = 1$

3.2 Random Forest

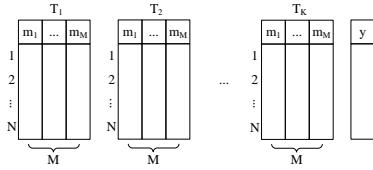
RndFor is an ensemble classifier, proposed by [Breiman, 2001], that takes advantage of randomness induced by splitting the instances and features for multiple classifiers. Decision trees are the classifiers and each of them receives a different training subset. The final classification output is the majority's decision of all the trees. That way, generalization error is reduced, impact of outliers and noise is minimized and model's performance is improved. For a training set that is defined according to previous Subsection 3.1 the RndFor classification algorithm is presented in Figure 1a. With T_i indicating an arbitrary tree and K as the number of trees, it contains following steps:

- (1) Assign each tree T_i with a subset of features of size between 1 and \sqrt{n} from the training set X
- (2) Take a bootstrap sample of instances from the training set (2/3 for **training** and 1/3 for **error estimation**)
- (3) Iteratively grow the trees using CART methodology without pruning, selecting the best feature and splitting the node into two daughters
- (4) Average the tree's error testing the subset of trees on **mutual error estimation subset** and testing each tree individually on its own **error estimation subset**
- (5) The output of classification is the majority vote of all trees in the forest

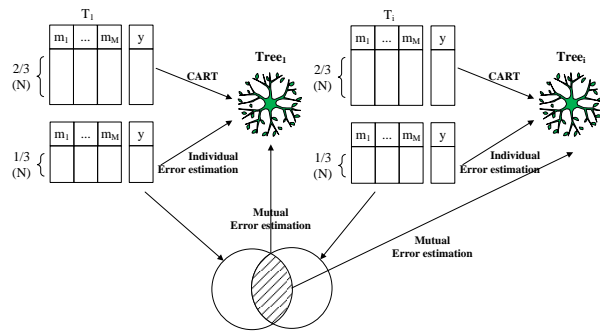
- The training set X contains n dependent variables (features) and y as dependent variable and N instances



- Each tree T_i (where $1 \leq i \leq K$) is given a random subset of features of size M (where $1 \leq M \leq n$)



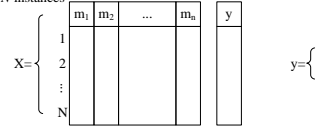
- Each tree T_i is given a bootstrapped subset of instances of size $2/3$ for learning and $1/3$ for error estimation
- CART methodology without pruning is used
- Error estimation is done individually and mutually for a subset of trees



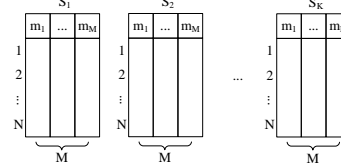
- Error is averaged across all trees
- Final classification decision is reached as a majority vote of all the trees

Fig. 1a Random Forest Algorithm

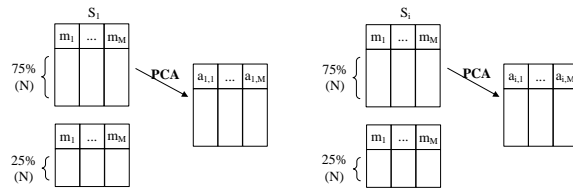
- The training set X contains n dependent variables (features) and y as dependent variable and N instances



- Each subset S_i (where $1 \leq i \leq K$) is given a random non-overlapping subset of features of size M (where $M = n/K$)



- Each subset S_i is reduced by randomly selected 25% of instances using bootstrap method
- PCA is applied to obtain coefficients a_{ij} (where $1 \leq i \leq K$ and $1 \leq j \leq M$) for each feature



- Organize the obtained coefficients from all subsets into a sparse "rotation" matrix R_k (where k is the index of future classifier)

$$R_k = \begin{bmatrix} [a_{1,1}, \dots, a_{1,M}] & \dots & [0] & \dots & [0] \\ [0] & \dots & [a_{i,1}, \dots, a_{i,M}] & \dots & [0] \\ [0] & \dots & [0] & \dots & [a_{k,1}, \dots, a_{k,M}] \end{bmatrix}$$

- Rearrange the coefficients in R_k so that they match the positions of the features they were constructed upon, yielding R_k^a
- Build k tree classifiers upon $(X \cdot R_k^a, Y)$ training dataset
- Final classification decision is reached as a majority vote of all the trees

Fig. 1b Rotation Forest Algorithm

3.3 Rotation Forest

The RttFor is a novel ensemble classifier algorithm, proposed by [Rodríguez et al., 2006]. It is an algorithm that involves randomized PCA performed upon a selection of features and instances of the training set before building the decision trees. For a training set that is defined according to previous Subsection 3.1, the RttFor classification algorithm is presented in Figure 1b. With S_i indicating an arbitrary subset of training set and K as the total number of subsets, it contains following steps:

- Split the features of training set X into K non-overlapping subsets of equal size $M=n/K$
- From each subset S_i , randomly remove 25% of instances using bootstrap method to ensure the coefficients obtained by PCA are different for each tree
- Run PCA on the remaining 75% of the subset and obtain a_{ij} coefficients for each i -th subset and j -th feature inside the subset
- Organize the a_{ij} coefficients in a sparse rotation matrix and rearrange the coefficients so that they match the positions of the n features they were built upon to obtain R_k^a
- Repeat the procedure in steps 1 – 4 for each tree in the RttFor and build the tree upon training set $X \cdot R_k^a, Y$
- The output of classification in the majority vote of all the trees in the forest

4. CASE STUDY

The goal of this case study is to evaluate the performance of RttFor in SDP. We used LogReg, a robust and trustworthy statistical classifier used in many other domains and the RndFor, a newer ensemble algorithm that is achieving excellent results in SDP, as the basis of our comparison. Our research question (**RQ**) is how well does RttFor perform in SDP when compared to LogReg and RndFor?

4.1 Datasets

We collected datasets from five consecutive releases of Eclipse JDT project: 2.0, 2.1, 3.0, 3.1 and 3.2. Eclipse is an integrated development environment (IDE), mostly written in Java, but offering a wide range of programming languages for development. The Java development tools (JDT) is one of the largest development environments and, due to publicly available source code management repository in GIT and bug tracking repository in Bugzilla, it is often analyzed in SDP research. The number of the source code files for each release of the JDT project and the ratio of NFPr and FPr files is given in Table I.

Table I. Eclipse Datasets

Dataset	Appearance		
	Files	NFpr	Fpr
JDT 2.0	2021	50.1%	49.9%
JDT 2.1	2343	64.1%	35.9%
JDT 3.0	2953	58.1%	41.9%
JDT 3.1	3394	64.5%	35.5%
JDT 3.2	1833	59.0%	41.0%

We collected the data using Bug Code (BuCo) Analyzer tool, which we developed for this purposes [Mauša et al., 2014]. The data collection approach starts with the collection of fixed bugs of severity greater than trivial, for each of the analyzed releases of the JDT project. Then it performs the bug-code linking on the file level of granularity using a regular expression that defines the surrounding characters of a Bug ID in the commit messages and outperforms even some of the complex prediction techniques [Mauša et al., 2014a]. Finally, it computes the 50 software product metrics using LOC Metrics and JHawk for each source code file. All the numerical metrics are used as independent variables for our experiment, so we excluded only the name of superclass. In order to make the dependent variable suitable for classification, the *number of bugs per file* was transformed into binary attribute named *Fault proneness*. Its value is set to 1 for all the files that contain at least 1 bug and set to 0 otherwise. The final structure of dataset is a matrix of size $N \times n$, where N represents the number of files and n represents the number of features.

4.2 Experiment Workflow

We used the Weka Experimenter Environment (version 3.6.9) to perform the experiment. Weka is a popular open source software for machine learning written in java, developed at the University of Waikato, New Zealand. The experiment workflow includes the following steps:

- (1) import the data in *arff* format and assign *fault proneness* to be the dependent variable
- (2) split the training and testing sets using 10-times 10-fold cross validation
- (3) build RttFor, LogReg and RndFor models
- (4) evaluate the models' performance in terms of Acc, TPR, FPR, F-measure, Kappa and AUC
- (5) perform the paired T-test of significance

The 10-times 10-fold cross validation is used to prepare the training and testing sets. The 10-fold cross validation divides the dataset into 10 parts of equal size, randomly picking instances (rows) and

maintaining the number of features (columns). In each of 10 steps, another 1/10 portion of dataset is used as testing set and the remaining 9/10 are used for training. The 10-times 10-fold cross validation iterates the whole procedure 10 times, evaluating the classification model for 100 times in total.

The classification models are all built and evaluated upon the same training and testing datasets, i.e. JDT release. It is important to be aware that the parameters tuning can improve the performance of various classifiers [Chen et al., 2009]. Thus, we performed an examination of their performance with various configurations. We used Weka's CVPParameter Selection meta classifier that performs parameter selection by cross-validation for any classifier. For RndFor, we were configuring the number of features per tree from 1 to 8 (8 is the default value, calculated as $\log_2(\text{number of attributes}) + 1$), the maximum depth of tree from 1 to 5 (including the default unlimited depth) and the number of trees from 10 to 50 with step of 5 (default value is 10). For RttFor, we were configuring only the number of groups from 3 to 10 (default value is 3) and left the number of iterations to default value of 10 and the percentage of removed instances to default value of 50%. Since there were no significantly different results obtained by either of these configurations, we left all the parameters to their default values for the main experiment.

4.3 Performance Evaluation

The evaluation of binary classification problems is usually done using the confusion matrix. Confusion matrix consists of correctly classified instances, true positive (TP) and true negative (TN), and incorrectly classified instances, false positive (FP) and false negative (FN). In our case, the Positive instances are the ones that are Fault Prone (FPr), i.e. files that have at least 1 bug, and Non-Fault-Prone files (NFPr) are the Negative ones. The evaluation metrics that we used in our experiment are the following ones:

- *Accuracy (Acc) – number of correctly classified files:*

$$Acc = \frac{TP+TN}{TP+FN+FP+TN} \quad (3)$$

- *True Positive Rate (TP_Rate, Recall) – number of correctly classified FPr files in total FPr files:*

$$TP\ Rate = \frac{TP}{TP+FN} \quad (4)$$

- *False Positive Rate (FP Rate) – number of incorrectly classified NFPr files in total false predictions:*

$$FP\ Rate = \frac{FP}{FP+FN} \quad (5)$$

- *F-measure (F) – harmonic mean of TPR and Precision:*

$$F = 2 \cdot \frac{TPR \cdot Precision}{TPR + Precision} \quad (6)$$

- *Kappa statistics – accuracy that takes into account the chance of random guessing:*

$$\kappa = \frac{Acc - Pr_{rand}}{1 - Pr_{rand}} \quad (7)$$

where Pr_{rand} is equal to:

$$Pr_{rand} = \frac{TP+FN}{Total} \cdot \frac{TP+FP}{Total} + \frac{TN+FP}{Total} \cdot \frac{TN+FN}{Total} \quad (8)$$

The usual output of a binary classifier is the probability that a certain instance belongs to the Positive class. Before making the final prediction, a probability threshold value, above which instances are going to be classified as Positive, needs to be determined. That is why all the above mentioned evaluation metrics are calculated with predetermined threshold value. A metric that does not depend on the threshold value is:

- *Area under receiver operating curve (AUC):*

$$AUC = \int_0^1 ROC_curve \quad (9)$$

where *ROC_curve* is a graphical plot that illustrates the relation between *TP_Rate* and *FP_Rate* for all possible probability threshold values.

All the used metrics have their values in range [0-1]. The performance of a classifier is better for higher values of Accuracy, TP Rate, F-measure, Kappa statistics and AUC. Only in the case of FP Rate, also known as the false alarm rate, the performance is better for lower values. It is important to use several evaluation metrics because they examine predictive performance from various angles. In the presence of severe data imbalance between the two output classes, it is even more important. For example, Acc can then easily become very high, misleading us to believe in excellent performance. On the other hand, the FP Rate would be also very high, indicating that the prediction is of questionable value [Mauša et al., 2012]. After obtaining the results, we perform the paired T-test in order to discover whether there are significant differences between classifiers. The whole process is repeated for each of the 5 Eclipse JDT datasets and for each of the 6 evaluation metrics.

5. RESULTS

The results of the paired T-tests for Acc, TPR, FPR, F-measure, Kappa statistics and AUC are given in Table II. Paired T-test compares only two classifiers at the time in one evaluation metric. First five rows represent the paired T-test comparison of results when the RttFor is the basis of comparison and the second five rows provide the results when the LogReg is compared to other two classifiers. The only remaining combination would be to use the RndFor as a comparison basis, but that one can be deduced from the previous two. The basis of comparison and its results are given in bold. The results are presented with their average value and standard deviation of 100 iterations that are obtained with the 10-times 10-fold cross validation process. The results that exhibit significant difference are marked with * and *v*. Sign * is given for cases in which the compared classifier is performing significantly worse than the basis of comparison and *v* is given for significantly better performance. The results that do not exhibit statistically significant difference have no sign adjacent to them. From results presented in tables II, we draw following observations:

- Overall summary of results shows that *RttFor* achieved 29 wins and 14 loses, *RndFor* achieved 19 wins and 14 loses, *LogReg* achieved 15 wins and 27 loses
- *RttFor* outperformed *RndFor* and *LogReg* in terms of AUC and Kappa statistics in all but 1 case.
- *LogReg* outperformed *RndFor* and *RttFor* in terms of FP rate in all but 1 case.
- *RttFor* is outperformed only in terms of TP Rate and FP Rate, 6 times by *RndFor* and 8 times by *LogReg*

5.1 Threats to Validity

It is very important to be aware of the threats to validity of our case study [Runeson and Höst, 2009] and we address them in this subsection. The generalization of our results is limited with the choice of the datasets we used. We covered only the evolution through 5 subsequent releases of only 1 project that comes from only 1 open source community. A greater number of case studies like this one, with as many datasets from various domains, is required to achieve a conclusion of greater confidence level. The choice of comparing classification algorithms is another threat to validity. This case study included only 2 classifiers other than RttFor. However, we chose the LogReg and the RndFor due to their very good performance in other case studies. The choice of classification model's parameters is a potential source of bias to our results. That is why we performed an analysis of performance when tuning the parameters. Since we noticed no statistically significant difference between various configurations, we left them to their default values.

Table II. Paired T-tests between Rotation Forest, Random Forest and Logistic Regression

Dataset	Accuracy			TP Rate			FP Rate		
	RttFor	RndFor	LogReg	RttFor	RndFor	LogReg	RttFor	RndFor	LogReg
JDT 2.0	0.75(0.03)	0.73(0.03) *	0.71(0.03) *	0.76(0.04)	0.77(0.04)	0.76(0.04)	0.26(0.04)	0.35(0.05) v	0.37(0.05) v
JDT 2.1	0.83(0.02)	0.82(0.02)	0.81(0.02) *	0.86(0.03)	0.88(0.03)	0.88(0.03)	0.38(0.06)	0.46(0.06) v	0.52(0.05) v
JDT 3.0	0.79(0.02)	0.79(0.02)	0.78(0.02) *	0.82(0.03)	0.84(0.03)	0.86(0.03) v	0.35(0.04)	0.40(0.04) v	0.48(0.04) v
JDT 3.1	0.82(0.02)	0.82(0.02)	0.81(0.01) *	0.87(0.03)	0.87(0.02)	0.91(0.02) v	0.43(0.05)	0.49(0.05) v	0.61(0.05) v
JDT 3.2	0.81(0.03)	0.80(0.02)	0.80(0.02)	0.82(0.04)	0.85(0.04) v	0.88(0.03) v	0.32(0.06)	0.38(0.06) v	0.47(0.06) v
	LogReg	RndFor	RttFor	LogReg	RndFor	RttFor	LogReg	RndFor	RttFor
JDT 2.0	0.71(0.03)	0.73(0.03)	0.75(0.03) v	0.76(0.04)	0.77(0.04)	0.76(0.04)	0.37(0.05)	0.35(0.05)	0.26(0.04) *
JDT 2.1	0.81(0.02)	0.82(0.02)	0.83(0.02) v	0.88(0.03)	0.88(0.03)	0.86(0.03)	0.52(0.05)	0.46(0.06) *	0.38(0.06) *
JDT 3.0	0.78(0.02)	0.79(0.02)	0.79(0.02) v	0.86(0.03)	0.84(0.03) *	0.82(0.03) *	0.48(0.04)	0.40(0.04) *	0.35(0.04) *
JDT 3.1	0.81(0.01)	0.82(0.02)	0.82(0.02) v	0.91(0.02)	0.87(0.02) *	0.87(0.03) *	0.61(0.05)	0.49(0.05) *	0.43(0.05) *
JDT 3.2	0.80(0.02)	0.80(0.02)	0.81(0.03)	0.88(0.03)	0.85(0.04) *	0.82(0.04) *	0.47(0.06)	0.38(0.06) *	0.32(0.06) *
	F-measure			Kappa			AUC		
	RttFor	RndFor	LogReg	RttFor	RndFor	LogReg	RttFor	RndFor	LogReg
JDT 2.0	0.75(0.03)	0.73(0.03) *	0.71(0.03) *	0.49(0.05)	0.42(0.06) *	0.38(0.06) *	0.83(0.03)	0.79(0.03) *	0.75(0.03) *
JDT 2.1	0.83(0.02)	0.82(0.02)	0.81(0.02) *	0.50(0.06)	0.44(0.06) *	0.39(0.06) *	0.84(0.02)	0.81(0.02) *	0.79(0.03) *
JDT 3.0	0.79(0.02)	0.79(0.02)	0.78(0.02) *	0.48(0.05)	0.45(0.04) *	0.40(0.05) *	0.82(0.02)	0.80(0.02) *	0.77(0.03) *
JDT 3.1	0.82(0.02)	0.82(0.02)	0.81(0.01) *	0.45(0.05)	0.41(0.05) *	0.34(0.05) *	0.81(0.02)	0.79(0.03) *	0.75(0.03) *
JDT 3.2	0.81(0.03)	0.80(0.02)	0.80(0.02)	0.51(0.07)	0.48(0.07)	0.43(0.06) *	0.84(0.03)	0.82(0.03) *	0.79(0.03) *
	LogReg	RndFor	RttFor	LogReg	RndFor	RttFor	LogReg	RndFor	RttFor
JDT 2.0	0.71(0.03)	0.73(0.03)	0.75(0.03) v	0.38(0.06)	0.42(0.06)	0.49(0.05) v	0.75(0.03)	0.79(0.03) v	0.83(0.03) v
JDT 2.1	0.81(0.02)	0.82(0.02)	0.83(0.02) v	0.39(0.06)	0.44(0.06) v	0.50(0.06) v	0.79(0.03)	0.81(0.02) v	0.84(0.02) v
JDT 3.0	0.78(0.02)	0.79(0.02)	0.79(0.02) v	0.40(0.05)	0.45(0.04) v	0.48(0.05) v	0.77(0.03)	0.80(0.02) v	0.82(0.02) v
JDT 3.1	0.81(0.01)	0.82(0.02)	0.82(0.02) v	0.34(0.05)	0.41(0.05) v	0.45(0.05) v	0.75(0.03)	0.79(0.03) v	0.81(0.02) v
JDT 3.2	0.80(0.02)	0.80(0.02)	0.81(0.03)	0.43(0.06)	0.48(0.07) v	0.51(0.07) v	0.79(0.03)	0.82(0.03) v	0.84(0.03) v

* – significantly worse, v – significantly better

6. RESULTS

This paper continues the search for a classification algorithm of utmost performance for the SDP research area. We analyzed a promising novel classifier called RttFor that received very limited attention in this field so far. We compared the performance of RttFor with the LogReg and the RndFor in terms of 6 diverse evaluation metrics in order to get as detailed comparison as possible. The classifiers were used upon 5 subsequent releases of Eclipse JDT open source project. These datasets contain between 1800 and 3400 instances, i.e. java files, 48 features describing their complexity and size and 1 binary class variable that indicates whether the file contains defects or not. The comparison was done using paired T-test of significance between results obtained by 10-times 10-fold cross-validation.

The conclusion of our case study and the answer to our **RQ** is that RttFor is indeed a state of the art algorithm for classification purposes. The overall ranking of the three classifiers we analyzed shows that the RttFor is the most successful one, the RndFor is the second best and the LogReg is the least successful classifier. This finding is consistent with other case studies that used RttFor and proves that this classifier needs to be taken into account when performing research in SDP more often. Our future work intentions include a more complex comparison that would include a greater number of classification

algorithms. But more importantly, it would include a greater number of datasets, covering longer periods of projects' evolution and a greater number of projects from various contexts.

REFERENCES

- T. Galinac Grbac, P. Runeson and D. Huljenic, 2013. A second replicated quantitative analysis of fault distributions in complex software systems. *IEEE Trans. Softw. Eng.* 39(4), pp. 462-476
- T. Galinac Grbac and D. Huljenic, 2014. On the probability distributions of faults in complex software systems, *Information and Software Technology* (0950-5849), pp. 1-26.
- J. J. Rodríguez, L.I. Kuncheva and C.J. Alonso, 2006. Rotation Forest: A New Classifier Ensemble Method, *IEEE Transactions on Pattern Analysis and Machine Learning*, vol. 28, no. 10, pp. 1619-1629.
- S. Lessmann, B. Basesens and S. Pietsch, 2008. Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings, *IEEE Transactions On Software Engineering*, vol. 34, no. 4, pp. 485-496.
- M. F. Amasyali and K. O. Ersoy, 2014. Classifier Ensembles with the Extended Space Forest, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 26, no. 3, pp. 549-562
- C. Pardo, J. F. Diez-Pasor, C. Garcia-Osorio and J. J. Rodriguez, 2013. Rotation Forest for regression, *Applied Mathematics and Computing*, vol. 219 (19), pp. 9914-9924.
- L. I. Kuncheva, J. J. Rodriguez, C. O. Plumpton, D. E. J. Linden and S. J. Johnston, 2010. Random Subspace Ensembles for fMRI Classification, *IEEE Transaction on Medical Imaging*, Vol. 29., No. 2, pp. 531-542
- J. Xia, P. Du, X. He and J. Chanussot, 2014. Hyperspectral Remote Sensing Image Classification Based on Rotation Forest, *IEEE Geoscience and Remote Sensing Letters*, Vol. 11., no. 1, pp. 239-243
- Bailing Zhang, 2013. Reliable Classification of Vehicle Types Based on Cascade Classifier Ensembles, *IEEE Transactions on Intelligent Transportation Systems*, Vol. 14, no. 1, pp. 322-332
- H. Palivela, H. K. Yogish, S. Vijaykumar and K. Patil, 2013. A study of mining algorithms for finding accurate results and marking irregularities in software fault prediction, *International Conference on Information Communication and Embedded Systems (ICICES)*, pp. 524–530.
- B. G. Tabachnick and L. S. Fidell, 2001. *Using Multivariate Statistics*, 5th edition, Pearson, 2007, ISBN 0-205-45938-2
- L. Breiman, 2001. Random forests, *Machine Learning*, Vol. 45, No. 1, pp. 5–32
- C. Chen, M.-L. Shyu and S.-C. Chen, 2009. Supervised Multi-Class Classification with Adaptive and Automatic Parameter Tuning, *IEEE International Conference on Information Reuse & Integration, IRI '09, Las Vegas, USA*, pp. 433-434
- G. Mauša, T. Galinac Grbac and B. Dalbelo Bašić, 2014. Software Defect Prediction with Bug-Code Analyzer – a Data Collection Tool Demo, *In Proceedings of SoftCOM '14. Split, Croatia*
- G. Mauša, P. Perković, T. Galinac Grbac and B. Dalbelo Bašić, 2014. Techniques for Bug-Code Linking, *In Proceedings of SQAMIA '14, Lovran, Croatia*, pp. 47-55
- G. Mauša, T. Galinac Grbac, and B. Dalbelo Bašić, 2012. Multivariate logistic regression prediction of fault-proneness in software modules, *In Proceedings of MIPRO '12, Opatija, Croatia*, pp. 698–703
- P. Runeson and M. Höst, 2009. Guidelines for conducting and reporting case study research in software engineering, *Empirical Softw. Engg.*, vol. 14, pp.131–164

Benefits of Using Domain Model Code Generation Framework in Medical Information Systems

PETAR RAJKOVIC, University of Nis, Faculty of Electronic Engineering

IVAN PETKOVIC, University of Nis, Faculty of Electronic Engineering

DRAGAN JANKOVIC, University of Nis, Faculty of Electronic Engineering

Both in medical information system development and upgrade project, we often face with a challenge of creating large number of reports and data collection forms. In order to reduce our efforts in this segment of system development, we tried to use various code generation and reporting tools. The main problem with standardized tools were lack of flexibility. Thus, we decided to develop domain model based framework that consists of data modeling, inverse engineering, code generation and model interpretation libraries and tools. Data modeling tool is used to create domain specific model starting from the loaded meta model. Both code generation and runtime interpretation tools use domain specific model as a basic input, and together with visual templates and generation/interpretation classes form easily extendable and customizable system. In our medical information systems development and upgrade projects we use both approaches and tend to define their proper roles in overall information system life cycle – from requirement collection phase to later system upgrades. In this paper we present basic building blocks of our framework and compare the effects of its usage against development when no automatic generation component is applied as well as when only standardized code generation tools are used. We managed to reduce development time in some segments of the system using domain model based generation tools to about one third of usually needed. The presented framework and its components are developed and tested during last six years and tested in four different development projects, around ten upgrades and in more than 25 information system deployment projects.

Categories and Subject Descriptors: **H.4.0 [Information Systems]:** General; **H.5.2 [User Interfaces]¹:** User interface management systems (UIMS); I.6.5 **[Simulation and modeling]:** Model development

General Terms: Human Factors, Design

Additional Key Words and Phrases: Model driven development, Code generation, Model interpretation

1. INTRODUCTION AND MOTIVATION

All major parts of information system's life cycle depend on a knowledge that come from many areas of computer sciences. At the same time, it requires the application of domain specific knowledge that should be incorporated into the system, in order to make developed software useful to its end users. Considering all levels of complexity, time needed for a system development is, pretty often, much longer than it is really necessary. Also, this period can be unacceptable for potential end users. The causes for this situation are very different, but in this paper we want to point one – time spend on a development of different components that share the same set of basic functionalities, but display different data – primarily data collection forms and reports.

In order to help developers solving this problem, software development environments offers different types of wizard-like tools that can load data model and then produce the form containing all required data entry fields and labels. Even though, these tools can support some complex views (master-detail or MVC), but when needed to be incorporated into the information system, developer must spend significant time to adapt their automatically generated logic before fitting the project. Furthermore, if generated forms need to support some translation mechanism, usually some time must be spent on this too. The similar story

This work is supported by the Ministry of Education and Science of Republic of Serbia (Project number #III47003).

Authors' addresses: Petar Rajkovic, University of Nis, Faculty of Electronic Engineering – Lab 534, Aleksandra Medvedeva 14, 18000 Nis, Serbia, e-mail: petar.rajkovic@elfak.ni.ac.rs; Ivan Petkovic, University of Nis, Faculty of Electronic Engineering – Lab 534, Aleksandra Medvedeva 14, 18000 Nis, Serbia, e-mail: ivan.petkovic@elfak.ni.ac.rs; Dragan Jankovic, University of Nis, Faculty of Electronic Engineering – Lab 105, Aleksandra Medvedeva 14, 18000 Nis, Serbia, e-mail: dragan.jankovic@elfak.ni.ac.rs;

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

In: Z. Budimac, M. Heričko (eds.): Proceedings of the 4th Workshop of Software Quality, Analysis, Monitoring, Improvement, and Applications (SQAMIA 2015), Maribor, Slovenia, 8.-10.6.2015. Also published online by CEUR Workshop Proceedings (CEUR-WS.org, ISSN 1613-0073)

can be told for reports. Even many reporting tools and engines can easily generate reports, their customization and later inclusion into the running software project can require significant amount of time.

The problem is not limited to the development phase, but also to later phases of upgrade and information system maintenance. Table 1 shows the statistic we extract from the project of medical information system (MIS) development for Primary and Ambulatory Care Center situated in Nis [Rajkovic et al 2009]. Nis is the biggest city in southern part of Serbia with population of about 250,000. Since it is a regional center, more than one and a quarter of a million of people from southern and eastern Serbia gravitates to our target institution for more complex medical examinations. The initial project has been running from 2009 till 2013, and during this period system was built incrementally. First departments started to use the system in second part of 2010 and system entered full service in all departments till the end of 2012. In this period, overall daily load of the system rose from about 3000 registered medical services daily to more than 13500. During 2013 and 2014 system was upgraded upon separate change requests. After the developed MIS was successfully deployed in Nis, the system was deployed in primary care centers of another 25 towns in southern and eastern Serbia. Furthermore, the extended and modified version was also used for Neurology Clinic in Nis supporting different set of initial system requirements [Milenkovic et al 2010].

Increased number of registered medical services was followed with the increase in number of requested reports and data collection forms. As it has been stated before, the primary care center has between 10 and 15 thousands of medical services (including examinations, therapeutic treatments and laboratory analysis) daily. Total number of different offered services is around 300, while around 250 are requested by government authorities. Many of them use standard data collection forms, but for significant number of services separate forms had to be developed. Next, medical personnel has to generate various reports and send them to Ministry of Health (MoH), insurance funds and other government agencies. Moreover, there are internal reports needed for covering internal business process as well as different medical research reports.

Table 1. Increase of the number of required reports and data collection forms per year

Calendar year	Medical examinations (daily)	External reports	Internal reports	Data collection forms	Required medical services
2011	2990	17	7	14	170
2012	7322	27	40	63	194
2013	13599	35	73	74	231
2014	13696	39	139	91	245

At the moment, our MIS dedicated for primary care facilities supports around 100 different data collection forms and almost 200 reports. Having in mind previous experience in information system development, and facing the constantly rising number of requested data collection forms and reports we tried two different approaches for making GUI development process more effective – use standard generation components that come with development environment and use domain model based custom built code generation and model interpretation tools. We are going towards a definition of a domain model-based framework that will help us when needed to automatically generate, test and deploy series of similar visual components. The framework, which was developed for initial project for Nis Primary Care Center, was intensively used to support specific requests for deployments in other medical facilities. The framework and its components were used in 4 development project (primary care MIS, hospital MIS, laboratory IS and radiology IS), around 10 upgrade and in more than 25 deployment projects.

In this paper we will presents the basic design elements of our framework and show its effect on the information system development process. Concepts of specific domain model creation, domain model extension and update are presented as basic building blocks of the framework. Furthermore automatic code generation and model interpretation are presented in following chapter. At the end, the comparison of results is presented and discussed.

2. RELATED WORK

The main elements of our model driven development framework are tool for domain specific data modeling, database structure check and inverse engineering library, code generation module and code interpretation library. Literature related to model driven engineering is pretty voluminous, but we can point out the paper [Meixner 2011] that elaborates and compares different approaches in model based GUI development along with past and current trends. Looking at the definitions, our approach could be categorized as holistic model driven GUI development process.

The base component of our framework is a modeling tool. It can be used for creation new and validation of existing models. Validation is based on design space exploration (DSE) principle and concepts based on dependency analysis described in [Hegedus et al 2011]. Our modeling tool is extended with reverse engineering library used to extract entities from legacy systems. In [Ramon 2014] the authors were focused on finding implicit layout and then generate appropriate GUI model. We follow this philosophy, but applied it on data definition level and the used generated model to generate not only GUI but various other components. We extended the inverse engineering tool with the component able to generate database administration application used for better understanding the structure of a legacy database. Our approach is based on environment named Teallach [Barclay et al 2003], which interface generation and environment customization concepts are partly reused. When working with legacy databases one problem that we often faced were vendor-specific functions and structures for databases. To overcome this problem we used the approach described in [El Akkaoui 2011]. This paper describes model driven framework that supports ETL (extract-transform-load) processes for data warehousing projects. The major part that we acquired were approach for creating vendor-independent sub-models and transformation based on a set on common interfaces.

Our code generation approach is based mostly on [Badreddin et al 2014], while widget based code interpretation was influenced by [Wanderman-Milne et al 2014] and [Ren et al 2010]. Comparing code generation and interpretation approach, the biggest drawback of interpreted component is lack of standardized test cases [Schlegel 2010], but the comparative advantage is their flexibility. Thus, we believe that both approaches have the right place in system development and can be used for different purposes.

3. DATA MODELING FRAMEWORK

Our data modeling framework is developed around initial meta model appropriate for MIS development. As a starting point we used OpenEHR meta model. Next, we adapted it and define extension points – entities that can be used as heading elements for newly defined items. Next step was to develop modeling tool that could be easily used both by developers and potential end users. The importance of the user friendly modeling tool is not only technical, but also project management related. This can help involving future users from the beginning of development process which eventually will lead to better results during system acceptance process.

The starting point in our project was to develop the extensions of domain specific model in a modeling tool. Modeling tool was used to define model extensions that will corresponds to future entities that will represent separate medical examinations or reports. Initially, we identified entities in starting model that can be further specialized (and call them extension points) and then use the modeling tool to define derived entities. The extension points can be configured including base entity name and specifying extension rules. These model extensions will be used as an input for generation tool that will be described in next section.

Another dimension to the modeling process is brought by legacy software and legacy data. In many cases, clients already have large amounts of data collected over the years which want to integrate into a new system. To complete the set of needed developing components, reverse engineering tool is a next that is needed (Fig. 1). Its main aim is to analyze existing data structures and automatically expand the model. Reverse engineering tool will initially load legacy data structure and examine it against existing meta model and extension point definition. Furthermore, reverse engineering tool can generate database administration application that helps in visualizing data structures in legacy database and helps in understanding relations between data entities.

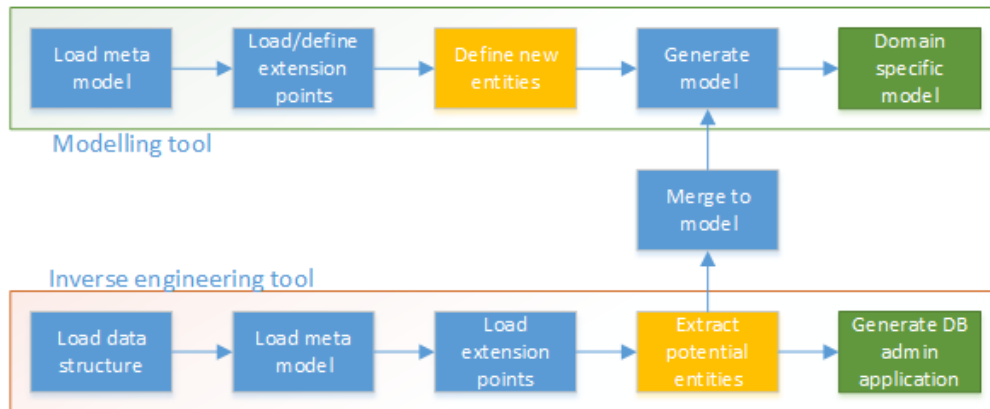


Fig. 1. Main functional block of data modeling and reverse engineering tool

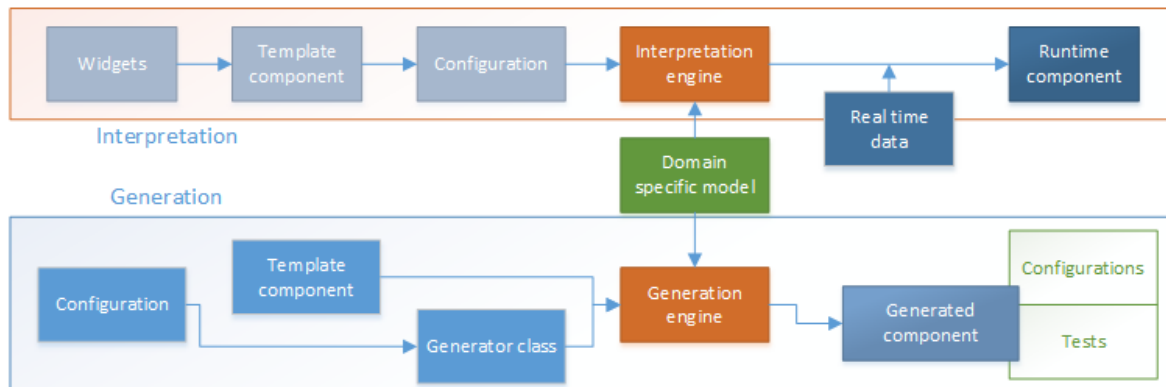


Fig. 2. Comparison of interpretation and generation processes based on domain specific model

When identify data tables that can satisfy extension point conditions, it will generate new entities that can be integrated in domain specific model. Before the extracted entities can be generated for domain specific model, they will be compared to the entities that already exist in order to find potential conflicts. Potential conflicts are examined on two cases, named Case A and Case B. Case A is a situation when two entities of the same name are found (one in the existing model, and another in the set of generated entities) – then the structure will be further examined. Case B is a situation when two entities of the same structure, but different names are found.

In case A, the tool will first check if the entities are defined under the same extension point. If so, user will be prompted with merging tool where can decide if should discard new version of the entity (extracted from legacy database), overwrite the existing entity definition with new version or pick which properties from the original and which from the final model should be included in a merge result. In case B, user must change the name of one of the entities to maintain naming convention in end model. At the moment there is a constraint that all entities must have different names.

Automatic conflict resolution can be enabled, and then it can be defined per extension point. Available strategies for merging in case A are:

- Keep existing entity. In this scenario changes from loaded entities will be ignored.
- Overwrite with loaded entity. Loaded entity will overwrite existing one.
- Merge properties. Properties that appear only in loaded entity will be added to existing entity.

In case B, only one automatic merge scenario is supported – change the name of loaded entity by adding the prefix that corresponds to extension point name.

Generated model is used then as a pivot element both for code generation and component interpretation engines (Fig. 2). Both of them use domain specific model as the main input and combine its structure with already prepared configurations and templates in order to produce final component.

4. COMPONENT GENERATION AND RUNTIME MODEL INTERPRETATION

Domain model, extension points and template components are used as the input for the next important step – automated software components generation. For this purpose specific highly customized generation tool has been developed. Generation tool loads domain model, template component, generator class and by using them creates a new software component that can be included in a software project or compiled and immediately used as a library [Rajkovic et al 2014]. Beside it is based on general approach, the system is optimized for Microsoft .NET platform.

Along with domain specific model entity, another base input for generation tool are template components. Our approach is to take already developed and tested software component, such as windows form, and then identify the parts of the code that can be used as general. Then, replace form specific parts with the specially marked comments that will be replaced during component generation process with a code generated on the base of the entity taken from the model. Since its generation process is based on templates, it has been used within our projects for automatic creation of several different classes of components – Windows forms, value selection components, translation resources and access privilege lists. Another benefit of this approach is improvement in component testing. For automatically generated component, automatic tests can be defined. For each of the fields and actions, set of predefined tests can be included. This will be the addition to the initial set of the tests loaded with chosen template component. After generation and testing, newly created components can be used for the extension of existing applications. On the base of mentioned model extension, and previously developed template software components (such are forms and reports), our generation tool will generate GUI elements that can be incorporated into information system project.

As it has been stated before, one big advantage in this approach is if users are involved from the initial stages of the project they can much easier accept the developed software later. This is not a new conclusion, even in a case study [Linberg 1999] the lack of communication with future user can lead to a project fall. Further analysis can be found in [Agarwal 2006] and [MacLeod et al 2011]. Also, we published our results and observation on this topic in [Rajkovic et al 2013].

Model based component generation is not the only way of domain model usage used within our framework during information system development process. The next approach is runtime model interpretation. For this purpose, special set of Web components is developed (Fig. 2).

The model interpretation library uses basically the same approach as the component generator – it loads entity from the model and the template in order to create a component that will be displayed in the browser. The main difference between this and the approach with code generation is that users can have two additional tools that can use online – configurator and the template definition tool. In the template definition tool users can define the visual elements of the displayed components – its arrangement, positioning, colors fonts etc. Configuration tool lets user further specify the appearance and the content of particular elements of the chosen template.

5. RESULTS AND DISCUSSION

Both presented approaches can be effectively used during information system development process. We had an opportunity to test them on many different types of components so we are able to present some relevant results and to define guidelines when to use which approach. Immediate effects that we get from our framework is reducing the time needed for component development (Table 2). In the table below, we presented the time needed for specific steps in component development process in cases when no optimization is used, then when we used only standard components and at the end when we used model driven approach through our framework. The data that we presented are gathered as a result of surveying our development team members, so they cannot be marked as fully accurate and objective, but they are indicative enough to compare the results in different approaches. We have interviewed twelve

developers currently involved in development projects that actively use our framework. Interviewed colleagues have at least two years long experience in information system development, while some of them work more than a decade in this field.

As the basic measurement unit we define T, which is a time that needed to define the structure of one entity and to create corresponding database table. All the other measurements are in correlation with mentioned T. After the creation of the database table, next step is defining a class in the object model. When using no optimization in development process, these two actions took the approximately the same time. But, in case when some object-relational model (ORM) is enabled in the project it is enough to create only one entity definition and it will be automatically used for creation both of the table and the entity.

Table 2. Comparison of time spent on developing single windows/web form using different approaches in development

Step	No optimization process applied	Using standard reports and component generators	Model driven dev.
Database table definition	T	0	0
Defining class in object model	T	T	T
Developing visual form	3T	0.1T	0.05T
Data validation methods implementation	3T	T	0.1T
Implementation of a logic specific for a form	2T	2T	2T
Defining configuration parameters	1T	1T	0.1T
Testing	6T	3T	0.5T
Overall time	15T	8.1T	3.75T

Next step is developing of visual form. It is related to pure creation of visual elements of the form and their connection to values retrieved by ORM classes. Since a developer needs to define label translations, data displaying components and to develop/inherit the logic to connect the form with the rest of the system we assumed that needed time is around 3T. When using some generation tools, we would get instantly created form and some simple adjustment is needed then. In case when we use our generation/interpretation tools this time is even shorter since no changes in visual style are usually required.

The component generation step is followed by implementation of data validation methods and the implementation of form-specific logic. While form-specific logic can hardly be replaced with automatically generated code (since it is a consequence of specific stakeholder requests) implementation of data validation method can be significantly tuned up. Standard component generators usually have check on the data type level, while our generation/interpretation tool can include also range checks that can be directly taken from used domain specific model.

Predominantly, our framework was used to create components for medical information systems, and many requirements specific for this area are included. We often have special requests to support data-field level configuration. In many cases the end users request the possibility to define which actions are possible for each field in some forms. Those requests lead to definition of specific configuration parameters. Depending on the number elements of the form, the process of defining and integrating configuration is process that lasts at least as the initial form definition. When our framework is used, we are able to define special generator class and template component so we could automatically generate configuration parameters at the same time while generating the form and therefore significantly reduce required time.

At the end of development process, testing and bug fixing lasted much longer when no generation component is used. Manual form building is process that many developers consider as less interesting and many different bugs came out when testing started. The most common problems are fields that are not connected to ORM properties and missing data validity checks. When using automated test generators, the time needed for this segment is halved. Using our framework, we managed to reduce this time significantly due to the fact that automatically generated and tested components are usually prone to mentioned errors. The segment of time that cannot be reduced is the one related to testing and fixing bugs from form-specific logic.

Table 3. Comparison the effects of different approaches in GUI components development on overall system development process

Category	Standard reports and component generators	Generated components	Interpreted components
Number of user sessions per module	4.33	2.76	2.08
Iterations before accepted solution	6.66	2.75	2.5
Mockup generation time	4 days	<1 day	<1 day
Average bug reports per module	9.65	3.74	3.35

The improvements related to visual components generation are not the only benefit we got. The side effect was improvement of overall development process (Table 3). Before start using our framework we relied our development process on standard report and component generation tools. Using modeling tool both by us and our customers reduced number of requirement collection session with users. Without modeling tool we have 4-5 sessions before final agreement per module. With modeling tool we reduced that number to around 2. Using interpreted components with some default templates made us possible that we can show initial form overview even during the first session. Also, using functionality from our reverse engineering tool, we are able to deliver application mockup for less than a day. Mockup application usually have uniform visual style and it is based on a single display template, but its main aim is to verify defined data structures and help arrange them logically. So, initial forms will have default view, but will be able to display all needed data with corresponding types and ranges. After initial session, we are focused on developing GUI templates in order to have proper preview of future functionalities. During second session with potential users we are usually able to demonstrate them GUI components mockup and to finalize stakeholder document in the sections related to user interface and required data structure.

Considering this, our users initially know what to expect, so when the project come to system acceptance and testing the number of change requests and bug reports is significantly lower if our framework is used. Comparing our two approaches – generated vs interpreted components – we can state that users have better response to Web solution based on interpreted components than to Windows interface based on generated forms. The main reason is the fact that the users do need to install any additional piece of software (they need only Web browser). Also, with installed template designer, the advanced users can develop their own templates and extend the existing system. On the other hand, the advantage of Windows forms based solution is faster response than Web applications.

After few years of system exploitation we realized that both approaches have their place in overall medical information system lifecycle. Windows applications are used by doctors and nurses in ordination and next to the medical instrumentation – in places when system needs to collect and process data. On the other hand, Web based components look like more desirable when data access, formatting and presenting is needed – in Web application offering medical record overview and in report generation.

6. CONCLUSION

This paper presents domain driven approach in automatic code generation process. To support this approach we have developed specific framework. The framework is actively used within our development team. We interviewed team members in order to get an estimate of the process improvement using the framework. Following their responses we can assume that the best effect we have in requirement collection phase where overall time needed is reduced to roughly 30-40%. When the project reaches development phase overall estimate based on mentioned survey is that the total time needed could be reduced to 25-30% of the time needed initially. The significant effect is visible on GUI based components, but core system components still need to be programed. When the project comes to deployment phase, the number of reported bugs is significantly lower than with the approach when no component generation tools are used. The main reason for this is that developers have initially generated forms with trusted functionalities, so they can focus primarily on the form's specific logic.

Our automatically generated/interpreted components helped not only development but also whole information system lifecycle, from initial mockup solution generation, through the development to later system upgrades and finally to deployment and maintenance. In all of these steps automatically

generated components reduced needed time. In combination with usage of modeling tools that improve communication with the customers and keep them involved during system development, they make eventual system acceptance easier and reduces number of customer reviews before final goal is reached.

REFERENCES

- Petar Rajković, Dragan Janković, and Aleksandar Milenković. 2013 "Developing and deploying medical information systems for Serbian public healthcare: Challenges, lessons learned and guidelines." *Computer Science and Information Systems* 10.3 (2013): 1429-1454.
- Petar Rajković, Dragan Janković, Aleksandar Milenković 2014, Improved Code Generation Tool for Faster Information System Development, SAUM 2014, Nis, Serbia, 12 - 14 November 2014, pp. 273 -276, Conference Proceedings, ISBN: 978-86-6125-117-7.
- Aleksandar Milenković; Petar Rajković; Dragan Janković; Tatjana Stanković; Miroslava Živković 2010: Software Module for Clinics of Neurology as a Part of Medical Information System Medis.NET, ICEST 2010, Ohrid, Makedonija, 23 - 26 Jun 2010; Proceedings, Vol. 1, br. 0, str. 323-326; ISSN: 978-9989-786-57-0;
- Petar Rajković; Dragan Janković; Tatjana Stanković 2009: An e-Health Solution for Ambulatory Facilities 2009, 9th International Conference on Information Technology and Applications in Biomedicine, ITAB 2009, Larnaca, Cyprus, November, 2009; Proceedings, Vol. 1, br. 1, str. Fr. 1.5.1 1-4; IEEE Publishing 2009, ISSN: 978-1-4244-5379-5
- Badreddin, Omar, Andrew Forward, and Timothy C. Lethbridge. 2014 "Improving Code Generation for Associations: Enforcing Multiplicity Constraints and Ensuring Referential Integrity." *Software Engineering Research, Management and Applications*. Springer International Publishing, 2014. 129-149.
- Wanderman-Milne, Skye, and Nong Li. 2014 "Runtime Code Generation in Cloudera Impala." *IEEE Data Eng. Bull.* 37.1 (2014): 31-37.
- Ramón, Óscar Sánchez, Jesús Sánchez Cuadrado, and Jesús García Molina. (2014) "Model-driven reverse engineering of legacy graphical user interfaces." *Automated Software Engineering* 21.2 (2014): 147-186.
- Linberg, Kurt R. "Software developer perceptions about software project failure: a case study." *Journal of Systems and Software* 49.2 (1999): 177-192.
- Agarwal, Nitin, and Urvashi Rathod. 2006 "Defining 'success' for software projects: An exploratory revelation." *International journal of project management* 24.4 (2006): 358-370.
- McLeod, Laurie, and Stephen G. MacDonell. 2011 "Factors that affect software systems development project outcomes: A survey of research." *ACM Computing Surveys (CSUR)* 43.4 (2011): 24.
- L. Ren, FengTian, X. Zhang and LinZhang, "DaisyViz 2010: A model-based user interface toolkit for interactive information visualization systems," *Journal of Visual Languages and Computing* 21, Elsevier, p. 209–229, 2010.
- P. Barclay, T. Griffiths, J. McKirdy, J. Kennedy, R. Cooper, N. Paton and P. Gray, 2003 "Teallach — a flexible user-interface development environment for object database applications," *Journal of Visual Languages & Computing*, Elsevier, vol. 14, no. 1, p. 47–77, 2003.
- Meixner, Gerrit, Fabio Paternò, and Jean Vanderdonck. 2011, "Past, Present, and Future of Model-Based User Interface Development." *i-com Zeitschrift für interaktive und kooperative Medien* 10.3 (2011): 2-11.
- T. Schlegel 2010, "An Interactive Process Meta Model for Runtime User Interface Generation and Adaptation," in *Fifth International Workshop on Model Driven Development of Advanced User Interfaces*, Atlanta, 2010.
- Zineb El Akkaoui, Esteban Zimanyi, Jose-Norberto Mazón, and Juan Trujillo. 2011. A model-driven framework for ETL process development. In *Proceedings of the ACM 14th international workshop on Data Warehousing and OLAP (DOLAP '11)*. ACM, New York, NY, USA, 45-52. DOI=10.1145/2064676.2064685 <http://doi.acm.org/10.1145/2064676.2064685>
- Abel Hegedus, Akos Horvath, Istvan Rath, and Daniel Varro. 2011. A model-driven framework for guided design space exploration. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE '11)*. IEEE Computer Society, Washington, DC, USA, 173-182. DOI=10.1109/ASE.2011.6100051 <http://dx.doi.org/10.1109/ASE.2011.6100051>

Towards the Formalization of Software Measurement by Involving Network Theory

GORDANA RAKIĆ, ZORAN BUDIMAC, MILOŠ SAVIĆ, MIRJANA IVANOVIĆ, University of Novi Sad

Complex network theory is based on a graph theory and statistical analysis. Software network is a sub-class of complex network and is usually represented by directed graphs representing relationships between software entities. Software metrics is a (numerical) measure that reflects some property of a software product or its specification. The goal of this paper is to set relationship between particular software metrics and corresponding measures from complex networks theory, including software networks. That relationship will help in discovering potentially new and useful metrics that are based on complex networks. Furthermore, it will narrow the gap between two similar research areas that is often too big. The specific goal of this paper is to present the method how the relationships can be established.

Categories and Subject Descriptors: **D.2.8 [Software Engineering]: Metrics - Complexity measures; Product metrics**

General Terms: Measurement, Standardization

Additional Key Words and Phrases: Software networks, Software metrics

1. INTRODUCTION

Real-world complex networks are used to model real-world and evolving systems in order to better understand them [Albert and Barabási 2002, Boccaletti et al. 2006, Newman 2003, Newman 2010]. They are most naturally represented as undirected or directed graphs denoted as $G = (N, L)$ or $G(N, K) = (N, L)$ or $G(N, K)$ or $G_{N, K}$ where:

- N is defined as $\{n_1, n_2, \dots, n_N\}$ - a set of nodes (vertices, points);
- L is defined as $\{l_1, l_2, \dots, l_K\}$ - a set of links (edges, lines), i.e., a set of pairs of elements of N ;
- N is a total number of nodes;
- K is a total number of links.

Besides representation in a form of graph, the complex network theory also provides a set of techniques for statistical analysis.

Software networks are a sub-class of complex networks and they are directed graphs representing relationships between software entities, .e.g. packages, classes, modules, methods, procedures, etc. They can be observed as a static representation of software code and design, and be used in analysis of the quality of software development process and software product with particular application in a field of large-scale software systems.

In software networks it can be differentiated between horizontal and vertical dimension. Horizontal dimension includes: low level (i.e., static call graph (SCG) /method collaboration network (MCN), function uses global variable (FUGV)), middle level (i.e., class/interface/module, etc. collaboration network, e.g. CCN - class collaboration network), and high level (i.e., package collaboration network). Vertical dimension usually reflects dependencies between software entities represented by hierarchy tree (e.g. package-class/interface-method/field).

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: G. Rakić, Z. Budimac, 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: {goca, zjb, svc, mira}@dmi.uns.ac.rs.

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

In: Z. Budimac, M. Heričko (eds.): Proceedings of the 4th Workshop of Software Quality, Analysis, Monitoring, Improvement, and Applications (SQAMIA 2015), Maribor, Slovenia, 8.-10.6.2015. Also published online by CEUR Workshop Proceedings (CEUR-WS.org, ISSN 1613-0073)

Software metric [Kan 2002] is defined as a function which maps some property of the software (e.g., whole software product, one piece of a software product, and specification of a software product) into a numerical value. The clear advantage of software networks is that they are based on well-understood, intuitive, and theoretically sound basis - graphs. The disadvantage is that the calculation of even the simplest measurements requires the construction of potentially huge graphs - see figure 1 for example [Savić et al. 2014]. The advantage of software metrics is that measurements can be often calculated in a simple way. The disadvantage is that they are based on various definitions and input data (e.g., source code, software architecture, graphs) and can often give different results for the same input data [Novak and Rakić 2010, Rakić and Budimac 2010]. For example, cyclomatic complexity [McCabe 1996], although precisely defined, can be calculated in at least two different ways: one by constructing the Control Flow Graph (CFG) and then finding linearly independent paths through it; and the second one: by counting decision points directly from the source code. For a characteristic example of an output of one software metrics tool see figure 2 [Bothe, 2012].

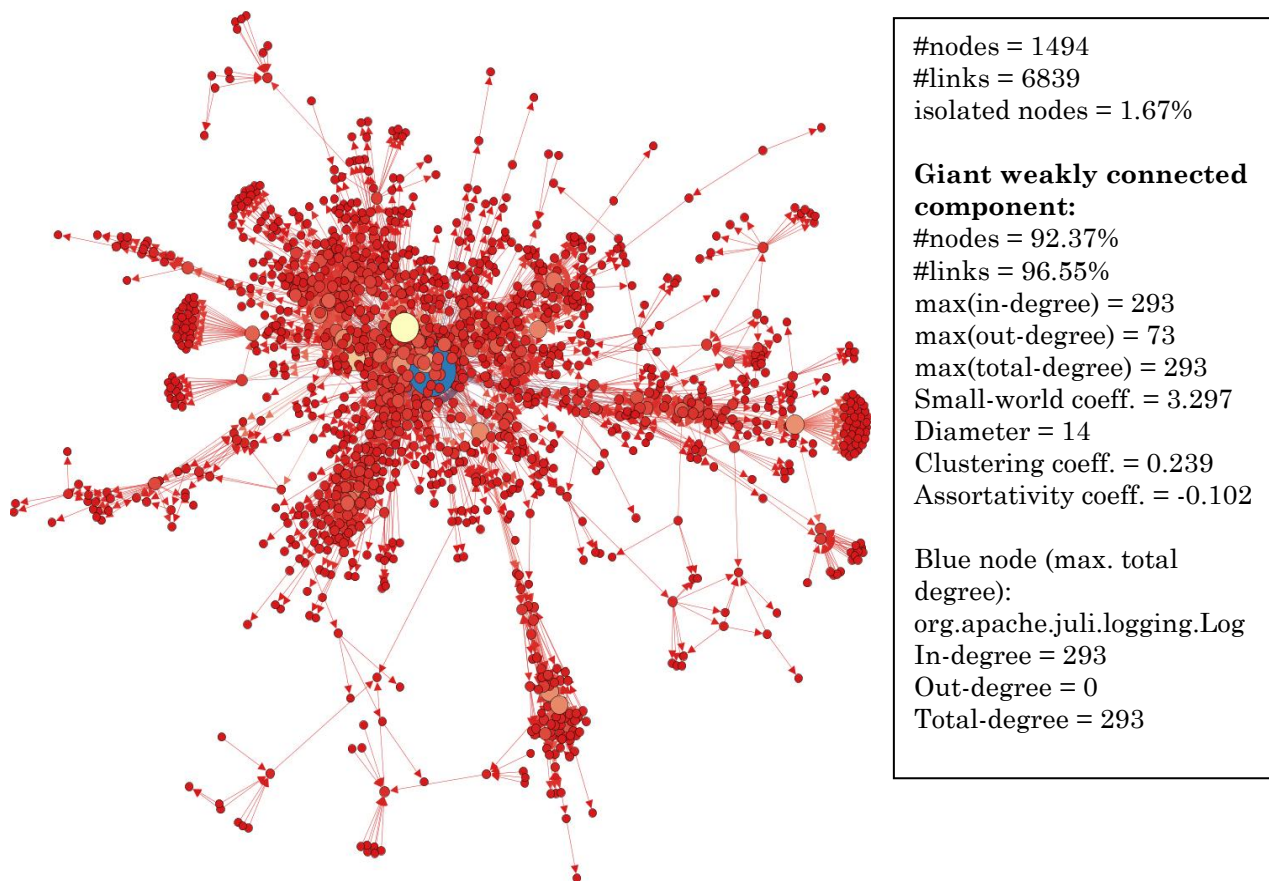


Fig. 1. Class collaboration network (CCN) (as part of a software network) for "tomcat-7.0.29 (java soft.)" with some numerical values.

Program: XCTL Unit Name	v(G)	ev(G)
TSteering::ParsingCmd(TCmdTag &, char *, char *, char *, char *)	70	24
DoCommandsFrame(HWND __*, unsigned int, long)	55	1
TAreaScan::CounterSetRequest(long)	45	10
TCalibrate::Dlg_OnCommand(HWND __*, int, HWND __*, unsigned int)	45	20
TAngleControl::Dlg_OnCommand(HWND __*, int, HWND __*, unsigned int)	45	6
DoCommandsChild(TMDIWindow __*, HWND __*, unsigned int, long)	43	4
TSteering::ParsingCmdParam(char *)	38	1
TTopographyExecute::Dlg_OnCommand(HWND __*, int, HWND __*, unsigned int)	38	7
TMList::ParsingAxis(char *)	37	34
TAreaScan::LoadMeasurementInfo(int)	35	30
TMotor::Initialize()	32	1
TAreaScan::SaveFile(int)	31	17
FrameWndProc(HWND __*, unsigned int, unsigned int, long)	30	13
TPlotData::DrawCoordinateSystem(HDC __*)	30	1
TBitmapSource::GenerateRLBitmap()	30	13
TMacroExecute::Dlg_OnCommand(HWND __*, int, HWND __*, unsigned int)	27	7
TAreaScan::LoadOldData(int)	26	19
TAreaScan::InitializeDlg(unsigned int, long)	25	5
TMain::TMain()	25	1
MenuSelect(HWND __*, unsigned int, long)	25	1
TAreaScan::InitializeTask(unsigned int, long)	25	11
TSetupStepScan::Dlg_OnCommand(HWND __*, int, HWND __*, unsigned int)	25	6
WndProc(HWND __*, unsigned int, unsigned int, long)	25	4
TCalibratePsd::Dlg_OnCommand(HWND __*, int, HWND __*, unsigned int)	23	1
TSetupAreaScan::Dlg_OnCommand(HWND __*, int, HWND __*, unsigned int)	23	5
TScan::LoadMeasurementInfo(int)	22	22
TBitmapSource::DrawMeasurementArea(HDC __*)	21	1
TBitmapSource::GenerateAngleSpaceBitmap(int, int, int, unsigned int)	21	5
TBraun_Psd::PsdReadOut(THowReadOutPsd)	19	1

Fig. 2. Values for Cyclomatic complexity (v(g)) for a program in experimental physics.

The goal of this paper is to set a method that would map one kind of measurements to the other. This would help to bring together researchers from both fields to hopefully adopt the common measurements, common definitions, and the common language between two research directions. Furthermore, due to a stronger theoretical background of software networks, it would achieve stronger formalisation of "classical" software metrics. Final consequence would be a stronger formal support to software quality control in general.

The rest of the paper is organized as follows. The second section overviews some of the most used metrics to illustrate similarities and differences between them. The third section describes our method of unifying several sets of metrics and gives a short example. Fifth section introduces related work, while the sixth one concludes the paper.

2. BACKGROUND

This section provides an overview of the most used software measurements, divided in several categories.

2.1 Network analysis measures

Measures frequently used in analysis of complex networks are related to the connectivity, distance, centrality, and clustering of nodes [Albert and Barabási 2002; Boccaletti et al. 2006; Newman 2003, 2010].

2.1.1 Connectivity

The most basic topological characteristic of a node is its degree – the number of links incident to the node. In the case of directed network we can distinguish between the in-degree (the number of in-coming links, fan-in) and the out-degree (the number of out-going links, fan-out) of a node. The connectivity of nodes in a regular network can be described by one number – the average degree. For a non-regular network the connectivity of nodes can be expressed by the degree distribution which is the probability $P(k)$ that randomly selected node has degree equal to k .

2.1.2 Distance

The distance between two nodes is defined as the length of the shortest path connecting them. A majority of real-world networks possess the *small-world* property. Having the small-world property means that the average distance between nodes in a network is a small value, much smaller than the number of nodes in the network. The harmonic mean of distances between nodes in the network reflects the communication efficiency of the network. The largest distance among nodes in a network is also known as the diameter of the network.

2.1.3 Centrality

Centrality measures rank nodes in a network with respect to their topological importance. The basic metrics of node importance are betweenness centrality, closeness centrality and eigenvector centrality. The betweenness centrality of a node is the extent to which the node is located on the shortest path connecting two arbitrary selected nodes. If the node is positioned on a large number of shortest paths then it has a vital role to the overall connectivity of the network. Consequently, nodes having high betweenness centrality are in position to maintain and control the spread of information across the network.

2.1.4 Clustering

A community or cluster in a network is a subset of nodes that are more densely connected among themselves than with the rest of the network. The quality of a partition of a network into communities is usually quantified by the Girvan-Newman modularity measure. The modularity measure accumulates the difference between the number of links in a community with the expected number of links among nodes constituting the community in a random network with the same degree distribution.

2.2 Software Metrics

Software metric can be defined as numerical value which reflects some property of a software development processes and software products [N. Fenton, 1996]. The mostly used categories of metrics are size, complexity and structure metrics, while structure metrics mainly reflect important aspects of product design. Classical software metrics are the oldest ones and the mostly used. They reflect size and complexity of source code. Structure is reflected by design metrics. This area of software metrics has expanded with development of object-oriented approach when different set of design metrics were introduced. Each of these sets usually contains also some newly introduced size and complexity metrics but they are always derived from some classical metric.

2.2.1 Classical size and complexity metrics

There are three mostly used classical software metrics:

- *Lines of Code (LOC)* family counts the lines of a program with or without comments, empty lines, etc.
- *Cyclomatic Complexity (CC)* counts linearly independent paths through a program
- *Halstead Metrics (H)* map complexity of a program to a number of operands and operators in it.

2.2.2 Design Metrics & Object Oriented Metrics

There are 4 generally accepted families of design and object oriented metrics. These families contain many intersections and there are still no universally accepted set of designing and object-oriented metrics.

- *Lorenz Metrics* [M. Lorenz, 1994] consist of :
 - *Average Method Size - AMS* based on LOC of every method.
 - *Average Number of Methods per Class – ANMC*
 - *Average Number of Instance Variables per Class - ANIVC*
 - *Class Hierarchy Nesting Level*
 - *Number of Subsystem-to-Subsystem Relationships - NSSR* is more general than CBO (see below)
 - *Number of Class-to-Class Relationships in Each Subsystem - NCCR* is analogous to CBO
 - *Instance Variable Usage - IVU*
 - *Average Number of Comment Lines (per Method) - ANM* based on LOC family of metrics
 - *Number of Problem Reports per Class*
 - *Number of Times Class is Reused*
 - *Number of Classes and Methods Thrown Away*

- *Morris metrics* [K. Morris, 1989] consist of :
 - *Methods per Class*
 - *Inheritance Dependencies*
 - *Degree of Coupling Between Objects* defined as total number of links / total number of objects
 - *Degree of Cohesion of Objects* = total number of incoming links / total number of objects
 - *Object Library Effectiveness* = total number of reusing object / total number of objects
 - *Factoring Effectiveness* = number of unique methods / total number of methods.
 - *Degree of Reuse of Inheritance Methods* is expressed as the percentage of really reused methods with respect to potentially reusable methods.
 - *Average Method Complexity* is based on CC metrics
 - *Application Granularity* also uses function point method for establishing the cost of a software project.

- *C&K metrics* [S. Chidamber and C. Kemerer. 1994] consist of
 - *Weighted Methods per Class (WMC)* is a sum of CC metrics for all methods of a class.
 - *Depth of Inheritance Tree (DIT)* is analogous to the *Class Hierarchy Nesting Level* (see above).
 - *Number of Children (NOC)* is the number of immediate sub-classes of a class.
 - *Coupling Between Object Classes (CBO)* is the number of other classes with which a class under consideration is connected.
 - *Response for a Class (RFC)* is a number of methods that can be called by the object of class under investigation.
 - *Lack of Cohesion in Methods (LCOM)* is the relationship between the number of methods in a class and the usage of variables in those methods.

- *MOOD metrics (Metrics for object oriented design)* [F. B. Abreu, 1995] consist of:
 - *Method Hiding Factor (MHF)* representing encapsulation
 - *Attribute Hiding Factor (AHF)* representing encapsulation
 - *Method Inheritance Factor (MIF)* representing inheritance
 - *Attribute Inheritance Factor (AIF)* representing inheritance
 - *Polymorphism Factor (PF)* representing polymorphism
 - *Coupling Factor (COF)* representing coupling

3. DEMONSTRATION OF THE MAPPING METHOD

We informally describe the method of uniting and harmonizing metrics from the two world in the following way:

- a) Take one complex network measure / software metrics measure,

- b) consider its possible meaning in software metrics measurements / complex network measurements,
- c) define relation,
- d) test relation if necessary.

For example,

- a) we can observe the node degree in a complex network (or in any directed graph), with an original meaning that an in-degree (fan-in) represents the number of incoming links into a node, while an out-degree (fan-out) represents the number of outgoing links from the node.
- b) semantics in the "software world" (e.g. dependency graphs or software architecture) can be easily found in (software) class collaboration network. If a class C is represented by a node then class references (in and out) are highly related to in- and out-degree in a complex network.
- c) Then we can set the following relations.

$$In(C) = (A \in In(C) \text{ iff } A \rightarrow C) \quad (1)$$

$$Out(C) = (A \in Out(C) \text{ iff } C \rightarrow A) \quad (2)$$

$$Fan-in(C) = |In(C)| \quad (3)$$

$$Fan-out(C) = |Out(C)| \quad (4)$$

$$CBO(C) = Fan-in(C) + Fan-out(C) - |In(C) \cap Out(C)| \text{ or} \quad (5)$$

$$CBO(C) = Fan-in(C) + Fan-out(C) \text{ iff } (In(C) \cap Out(C)) = \emptyset \quad (6)$$

where: $a \rightarrow b$ denotes that there is a link from a to b , $In(C)$ is the set of nodes that references C , $Out(C)$ is the set of nodes referenced by C and $CBO(C)$ is "Coupling Between Object Classes" metrics from C&K set of metrics.

d) Relations that are presented in this paper are formally provable (or understandable from their basic explanations), therefore the (statistical) testing on these relations is not necessary. Otherwise, a statistical proof should be provided by using any tools available, but the SSQSA framework (Set of Software Quality Static Analyzers) [G. Rakić et al., 2013] would provide the most reliable results because it is based on a common language- and metrics-independent internal structure. Furthermore, SSQSA already implements many of described metric algorithms, therefore it is easy to run most of the needed statistical test.

4. RELATED WORK

In [M. Lanza and R. Marinsku, 2006] a numerous observations using classical and object-oriented metrics are given. However, no relations to complex (software) networks are given. [L. Šubelj and M. Bajec, 2012] discussed how some of the features and measures in general complex networks can be used to reason about the quality of software. For example, the nodes (software entities) with high betweenness are influential and an extra care should be paid to their maintenance. Similarly software entities with high $Fan-in$ values are often used and are thus candidates for reuse, while those with high $Fan-in$ are complex ones. However, no relationships between those and other (classical and object-oriented) metrics were discussed.

5. CONSLUSION AND FURTHER WORK

Our goal is to join two researches in the field of general software measurements: "classical" software metrics (incl. object-oriented one) and complex (software) networks. Many metrics in both fields are analogous to each other, some are the same (but with different names and different description mechanisms) and some are (at the first sight) unrelated (figure 3).

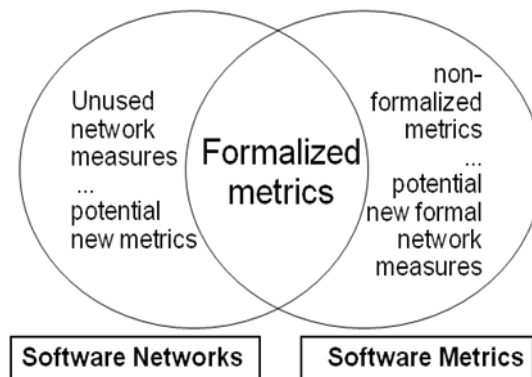


Fig. 3. The relationship between two kinds of software measurements.

By using the method that we proposed in this paper we hope to establish a solid and formalized intersection of both metrics. In doing so we are based on (software) networks that already have a formal background, while "classical" metrics are often described textually. Additional contribution may be achieved after mapping of all possible measures. The rest of measure may potentially be defined as new software metric derived from network measure, or wise versa.

REFERENCES

- F. B. Abreu. 1995. Design Quality Metrics for Object-Oriented Software Systems. ERCIM News No.23 - October 1995 - INESC. http://www.ercim.org/publication/Ercim_News/enw23/abreu.html
- R. Albert and A.-L. Barabási. 2002. Statistical mechanics of complex networks, *Rev. Mod. Phys.* 74 (1), pp. 47–97, DOI:<http://dx.doi.org/10.1103/RevModPhys.74.47>.
- S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, D. Hwang. 2006. Complex networks: structure and dynamics, *Phys. Rep.* 424 (45), pp. 175–308, DOI:<http://dx.doi.org/10.1016/j.physrep.2005.10.009>.
- K. Bothe. 2012. Lecture notes for Software engineering course, Humboldt University Berlin.
- S. Chidamber and C. Kemerer. 1994. A Metrics Suite for Object Oriented Design, *IEEE Trans. Software Eng.*, Vol 20, no 6, June, pp. 476-493.
- N. Fenton, S. L. P. (1996). *Software Metrics: A Rigorous and Practical Approach*. Thomson Computer Press
- M. Lanza and R. Marinescu. 2006. *Object-oriented metrics in practice*, Springer, Berlin, ISBN10 3-540-24429-8.
- M. Lorenz. 1994. *Object-oriented software metrics: a practical guide*, Prentice Hall, USA, ISBN:0-13-179292-X.
- S. H. Kan. 2002. *Metrics and models in software quality engineering*, Addison-Wesley Longman Publishing Co., Inc., 2002.
- J. McCabe. 1976. A Complexity Measure, *IEEE Transactions on Software Engineering*, pp. 308–320.
- K. Morris. 1989. *Metrics for object-oriented software development environments*, Sloan School of Management, MIT, Boston, USA, 135 pages.
- M.E.J. Newman. 2003. The structure and function of complex networks, *SIAM Rev.* 45, pp. 167–256, DOI:<http://dx.doi.org/10.1137/S003614450342480>.
- M. E. J. Newman. 2010. *Networks: An Introduction*, Oxford University Press Inc., New York, NY, USA, 2010.
- J. Novak and G. Rakić. 2010. Comparison of software metrics tools for .NET. In *Proc. of 13th International Multiconference Information Society-IS*, Vol A, pages 231–234, 2010.
- G. Rakić and Z. Budimac. 2010. Problems in systematic application of software metrics and possible solution. In *Proc. of The 5th International Conference on Information Technology (ICIT)*, Amman, Jordan, 2010.
- G. Rakić, Z. Budimac, Z., and M. Savić, 2013. Language independent framework for static code analysis. In *Proc. of the Balkan Conference in Informatics 2013*, BCI '13, Thessaloniki, Greece, September 19-21, 2013, pp. 236–243.
- 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* 56, Elsevier, DOI:10.1016/j.infsof.2014.04.011, pp.1268-1288.
- L. Šubelj and M. Bajec. 2012. Software systems through complex networks science: Review, analysis and applications, In *Proc. of First International Workshop on Software Mining*, ACM, pp. 9-16.

Validation of Static Program Analysis Tools by Self-Application: A Case Study

MILOŠ SAVIĆ AND MIRJANA IVANOVIĆ, University of Novi Sad

The validation of static program analysis tools is an extremely hard and time consuming process since those tools process source code of computer programs that are usually extremely large and complex. In this paper we argue that static program analysis tools can be validated by self-application, i.e. by applying a source code analysis tool to its own source code. Namely, developers of a complex source code analysis tool are familiar with its source code and consequently in position to more quickly examine whether obtained results are correct. The idea is demonstrated by the application of SNEIPL, a language-independent extractor of dependencies between source code entities, to itself.

Categories and Subject Descriptors: D.2.4 [Software Engineering]: Software/Program Verification—*Validation*; D.2.7 [Software Engineering]: Distribution, Maintenance, and Enhancement—*Restructuring, reverse engineering, and reengineering*

General Terms: Experimentation, Measurement

Additional Key Words and Phrases: Source code analysis, self-application, validation

1. INTRODUCTION

Static program analysis tools process source code of computer programs in order to extract information that can help software engineers in a variety of tasks ranging from program understanding to fault detection [Binkley 2007]. The automated extraction of information in static program analysis is done without executing program and relies only on source code or some intermediate representation. Software validation refers to the process of evaluation of a software system in order to check whether it works properly and according to its specification. The validation of static program analysis tools is an extremely important task since those tools are used to understand and improve software systems. On the other hand, real-world software systems are usually extremely large and hard to comprehend making the validation hard and time consuming.

The identification of dependencies between source code entities (functions, classes, modules, etc.) is one of fundamental problems in static program analysis. We use the generic term “software network” to denote directed graphs of dependencies between source code entities. The importance of software networks extraction spans multiple fields such as empirical analysis of complexity of software systems, their reverse engineering and computation of software design metrics [Savić et al. 2014]. In our previous works [Savić et al. 2012b; 2014] we introduced SNEIPL – a language-independent approach

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. The authors would like to thank professor Zoran Budimac for valuable comments on an early version of this paper.

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, M. Heričko (eds.): Proceedings of the 4th Workshop of Software Quality, Analysis, Monitoring, Improvement, and Applications (SQAMIA 2015), Maribor, Slovenia, 8.-10.6.2015. Also published online by CEUR Workshop Proceedings (CEUR-WS.org, ISSN 1613-0073)

to the extraction of software networks representing software systems at various levels of granularity¹. In the experiment described in [Savić et al. 2012b] we showed that SNEIPL extracts isomorphic networks from simple, but structurally and semantically equivalent software systems written in different programming languages. In the subsequent research [Savić et al. 2014] we demonstrated that SNEIPL is able to extract software networks from real-world software systems written in different programming languages (Java, Modula-2 and Delphi). Moreover, we showed that class collaboration networks extracted from 10 real-world Java software systems are highly similar to those obtained by a language-dependent tool which forms networks from byte code, and much precise than the networks obtained by a language-independent tool which employs a lightweight, fuzzy parsing approach to dependency extraction. The research presented in this paper follows previously conducted experiments related to the validation of SNEIPL. More specifically, we used SNEIPL to extract software networks from its own source code. Due to the familiarity with the implementation of SNEIPL we were in position to quickly determine whether obtained results are correct.

The rest of the paper is structured as follows. Related work is presented in Section 2. The short description of the tool is given in Section 3. Experiments and results are presented in Section 4. The last Section concludes the paper.

2. RELATED WORK

There is a large number of tools that identify dependencies between source code entities. Usually they are language-dependent (tied to a particular programming language). For example, the overview of existing static call graphs extractors for C/C++ can be found in [Murphy et al. 1998; Telea et al. 2009]. Software networks can also be extracted from software documentation such as JavaDoc HTML pages (up to a certain level of precision) or low-level intermediate representations such as Java byte code. Software networks extractors rely either on traditional parsing techniques or employ more lightweight, but less precise, approaches based on pattern matching [Kienle and Müller 2010].

In research works that deal with the analysis of software systems under the framework of complex network theory software networks are usually extracted using language-specific tools:

- in [Valverde and Solé 2007; de Moura et al. 2003] analyzed networks were extracted by lightweight, handmade parsers of C/C++ header files,
- in [Jenkins and Kirk 2007; Louridas et al. 2008] by tools that rely on Java bytecode parsers,
- in [Wang et al. 2013] by parsing C source code using a modified version of the GCC compiler,
- in [Taube-Schock et al. 2011] by extending the standard Java parser of the Eclipse IDE,
- in [Wheeldon and Counsell 2003] by using Java Doctlet capabilities to inspect the source code structure,
- in [Puppini and Silvestri 2006] by parsing JavaDoc HTML pages,
- in [Savić et al. 2011; Savić et al. 2012a] by a tool that relies on Java parser generated by JavaCC.

The identification of dependencies among source code entities in existing language-independent reverse engineering tools can be classified into two categories:

- Tools that have separate fact extractors (source code models in terms of software networks) for each supported language. Examples of such tools are Rigi [Kienle and Müller 2010], GUPRO [Ebert et al. 2002] and Moose [Ducasse et al. 2000].

¹The tool can be downloaded at <https://code.google.com/p/ssqsa/>

—Tools that realize partially language-independent fact extraction. This means that for a subset of supported languages software networks are formed from a low-level (statement-level) language-independent source code representation, while for other supported languages there are separate fact extractors. An example of a tool that belongs to this category is Bauhaus [Raza et al. 2006].

The validation of dependency extraction in aforementioned tools was conducted on several real-world computer programs, none of them being the reverse engineering tool itself.

3. SNEIPL TOOL

SNEIPL has been implemented as one of the back-ends of the SSQSA framework [Rakić et al. 2013; Budimac et al. 2012]. The whole SSQSA framework is organized around the enriched Concrete Syntax Tree (eCST) representation of source code [Rakić and Budimac 2011a; 2011b] that is produced by the SSQSA front-end known as eCSTGenerator. The eCST representation is a language-independent source code representation and makes SSQSA back-ends independent of programming language. The concept of universal nodes introduced in the eCST representation is what makes it substantially different from other tree representations of source code. Namely, eCST universal nodes are predefined language-independent markers of semantic concepts expressed by concrete language constructs. One universal node in an eCST denotes particular semantic concept realized by the syntax construction embedded into the eCST sub-tree rooted at the universal node.

From an input set of eCSTs SNEIPL forms a heterogeneous software network that is known as *General Dependency Network* (GDN). GDN shows dependencies among software entities reflecting the design structure of a software system [Savić et al. 2014]. Nodes in a GDN represent architectural elements of a software system: packages, classes/modules, interfaces, functions/methods and global variables/class attributes. GDN links represent various types of relations: CALLS relations between functions, REFERENCES relations between package-level entities, REFERENCES relations between class-level entities, relations that represent different forms of class coupling, USES relations between functions and variables, and CONTAINS relations that reflect the hierarchy of entities.

The set of eCST universal nodes, among others, contains entity-level universal nodes which mark definitions/declarations of software entities. SNEIPL deduces vertical dependencies (CONTAINS relations) from the hierarchy of entity-level eCST universal nodes in input eCSTs. Calls relations between functions are recognized by analysis of sub-trees rooted at the FUNCTION_CALL universal node which marks function calls. Relations among class-level entities are identified by analysis of sub-trees rooted at the TYPE universal node which marks type identifiers. Finally, relations between functions and global variables are identified by analysis of sub-trees rooted at the NAME universal node which marks all identifiers present in the source code. To match an identifier (name of variable, type or invoked function) with its definition SNEIPL uses the name resolution algorithm that is based on several components: previously identified vertical dependencies, information contained in import statements (statements that are marked with the IMPORT_DECL universal node), information contained in local symbol tables that are attached to FUNCTION_DECL (marks function definitions) and BLOCK_SCOPE (marks block of statements) universal nodes, lexical scoping rules and rapid type analysis [Bacon and Sweeney 1996] that is adopted for the eCST representation. The more detailed description of the name resolution algorithm is given in [Savić et al. 2014].

4. EXPERIMENT AND RESULTS

SNEIPL has been written in the Java programming language. The implementation of SNEIPL consists of 6876 lines of code (without empty lines) which means that SNEIPL is a non-trivial software system of moderate size. Using eCSTGenerator we transformed the source code into the eCST representation.

The eCST representation of SNEIPL consists of 54 eCSTs (one eCST correspond to one compilation unit). Then we employed SNEIPL to extract software networks from the SNEIPL implementation. Obtained GDN has 584 nodes and 2285 links.

4.1 Recovery of SNEIPL's architecture

Package collaboration networks (PCN) show dependencies among packages and thus represent the architecture of software systems at the highest level of abstraction. Figure 1 shows the SNEIPL PCN extracted using SNEIPL. Due to the familiarity with the implementation of SNEIPL we were in position to extremely quickly validate that the extracted PCN is actually correct.

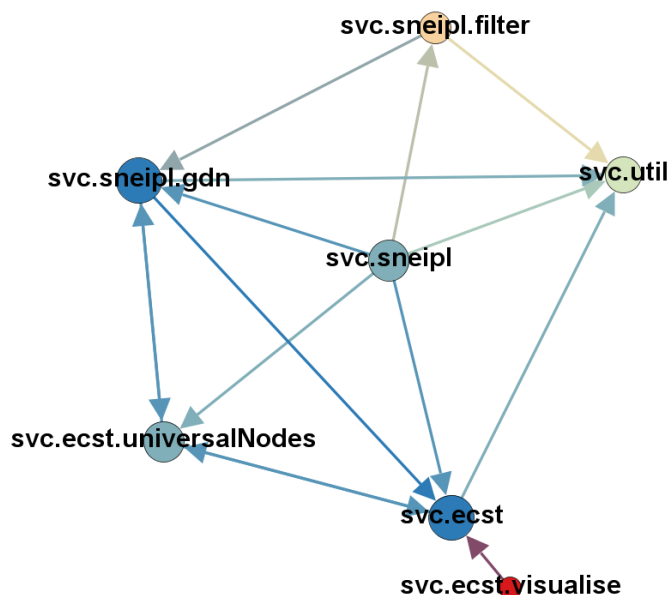


Fig. 1. The package collaboration network of SNEIPL.

The package **svc.sneipl** encompasses core SNEIPL's classes – classes which identify software entities and dependencies between them relying on input eCSTs. This package depends on all other SNEIPL packages except the **svc.ecst.visualise** package. **svc.ecst.visualise** contains source code of a simple and standalone GUI application that visualizes eCST trees. We mainly used it during the development and testing of SNEIPL when it was necessary to have input XML files visually observable. In contrast to **svc.sneipl**, **svc.util** groups simple utility classes. Therefore, this package does not depend on any other package.

The package **svc.ecst** contains classes which provide functionalities related to the eCST representation. The eCST representation of one compilation unit is internally stored as an object of **svc.ecst.-ECSTTree** class. This class defines methods which load eCST from an input XML file produced by eCSTGenerator and holds the reference to the root node. Each node in loaded eCST is represented as an object of **svc.ecst.ECSTNode** class which holds the content of the node, the reference to the parent node and the list of references to child nodes. The package **svc.ecst.universalNodes** groups classes that represent different eCST universal nodes used by SNEIPL. One eCST universal node is represented by a class that directly or indirectly extends **svc.ecst.ECSTNode** class. Class

svc.ecst.ECSTTree makes objects representing universal nodes and consequently **svc.ecst** depends on **svc.ecst.universalNodes**. The dependency between **svc.ecst** and **svc.ecst.universalNodes** is reciprocal because universal nodes are instances of **svc.ecst.ECSTNode**.

SNEIPL recognizes software entities and dependencies among them according to the language-independent procedure that relies only on eCST universal nodes. Thus, the core package depends on **svc.ecst.UniversalNodes**. When an entity/dependency is recognized the appropriate node/link in the GDN is created. Thus, the core package depends on **svc.sneipl.gdn**. Each GDN node corresponds to one eCST universal node marking the definition/declaration of a software entity and consequently **svc.sneipl.gdn** depends on **svc.ecst** and **svc.ecst.universalNodes**. The dependency between **svc.sneipl.gdn** and **svc.ecst.universalNodes** is reciprocal since local symbol tables can be attached to some universal nodes. Local symbol tables contain local variables defined in functions and block statements. Those variables can have types that correspond to GDN nodes and consequently **svc.ecst.universalNodes** depends on **svc.sneipl.gdn**.

Finally, the main SNEIPL class contained in the core package instantiates filters from **svc.sneipl-filter** to isolate specific software networks from the formed GDN. Thus, the core package depends on the filter package, while the filter package depends on the gdn package.

4.2 Isolated entities

Isolated nodes in extracted software networks can indicate missing links, and thus can point to errors in the implementation of software networks extraction tool. Therefore, we determined and examined characteristics of isolated nodes in the networks representing SNEIPL. The package and class collaboration network of SNEIPL do not contain isolated nodes (unused packages and classes). On the other hand, isolated nodes can be observed in the SNEIPL static call graph (SCG) and FUGV (Function Uses Gloval Variable) network.

The SNEIPL SCG contains 34 isolated nodes (10.36% of the total number). Table I shows the list of isolated nodes. To each method SNEIPL assigns a name that can be described by the following regular expression:

$$F'?' M ('@' T)^* '#' R,$$

where F denotes the fully qualified name of a class/interface which declares/defines the method, m is the name of the method, T denotes the type of a formal argument of m , while R is the return type of m . Seven methods listed in the table are methods that are unused – methods that are never called by other methods, nor they call other methods defined in the SNEIPL source code. Those methods can be safely removed from the SNEIPL source code distribution. Three isolated nodes represent method declarations from *SymTab* – the only interface defined in SNEIPL. The *SymTab* interface is implemented by classes representing eCST universal nodes to which local symbol tables can be attached. Those three nodes are isolated simply because in-coming links are given to the nodes representing implementations of those declarations in classes that implement the *SymTab* interface. Three isolated methods listed in Table I are so called call-back methods – methods defined in the SNEIPL source code that are called only from methods contained in external frameworks. Namely, SNEIPL defines four transform methods that are called by the JUNG library to export extracted software networks in the Pajek network file format. One of those methods calls one method defined in SNEIPL, and consequently it is represented by a non-isolated node in the SNEIPL SCG. Other three transform methods do not rely on SNEIPL methods and consequently they are isolated nodes. One method listed in the table is GUI method – method that is activated when a button is clicked in the GUI application that visualizes eCST trees.

Table I. Isolated nodes in the SNEIPL static call graph.

Method name	Explanation
svc.ecst.ECSTNode.disposeChilds#void	Unused
svc.ecst.ECSTNode.emptySubtree#boolean	Unused
svc.ecst.ECSTNode.findAllAtFirstLevel@String#LinkedList	Unused
svc.ecst.ECSTNode.findFirstAtFirstLevel@String#ECSTNode	Unused
svc.ecst.ECSTNode.rewriteToken@String#void	Unused
svc.ecst.ECSTTypedNode.ECSTTypedNode@String@ECSTNode@boolean@boolean#void	Called only via Reflection API
svc.ecst.universalNodes.Argument.Argument@String@ECSTNode#void	Called only via Reflection API
svc.ecst.universalNodes.ArgumentList.ArgumentList@String@ECSTNode#void	Called only via Reflection API
svc.ecst.universalNodes.AttributeDecl.AttributeDecl@String@ECSTNode#void	Called only via Reflection API
svc.ecst.universalNodes.BlockScope.BlockScope@String@ECSTNode#void	Called only via Reflection API
svc.ecst.universalNodes.ConcreteUnitDecl.ConcreteUnitDecl@String@ECSTNode#void	Called only via Reflection API
svc.ecst.universalNodes.Extends.Extends@String@ECSTNode#void	Called only via Reflection API
svc.ecst.universalNodes.FormalParamList.FormalParamList@String@ECSTNode#void	Called only via Reflection API
svc.ecst.universalNodes.FunctionCall.FunctionCall@String@ECSTNode#void	Called only via Reflection API
svc.ecst.universalNodes.FunctionDecl.FunctionDecl@String@ECSTNode#void	Called only via Reflection API
svc.ecst.universalNodes.Implements.Implements@String@ECSTNode#void	Called only via Reflection API
svc.ecst.universalNodes.ImportDecl.ImportDecl@String@ECSTNode#void	Called only via Reflection API
svc.ecst.universalNodes.Instantiates.Instantiates@String@ECSTNode#void	Called only via Reflection API
svc.ecst.universalNodes.InterfaceUnitDecl.InterfaceUnitDecl@String@ECSTNode#void	Called only via Reflection API
svc.ecst.universalNodes.Name.Name@String@ECSTNode#void	Called only via Reflection API
svc.ecst.universalNodes.PackageDecl.PackageDecl@String@ECSTNode#void	Called only via Reflection API
svc.ecst.universalNodes.ParameterDecl.ParameterDecl@String@ECSTNode#void	Called only via Reflection API
svc.ecst.universalNodes.SymTab.getType@String#GDNNode	Interface declaration
svc.ecst.universalNodes.SymTab.getTypeAsStr@String#String	Interface declaration
svc.ecst.universalNodes.SymTab.nameDeclaredHere@String#boolean	Interface declaration
svc.ecst.universalNodes.Type.Type@String@ECSTNode#void	Called only via Reflection API
svc.ecst.universalNodes.TypeDecl.TypeDecl@String@ECSTNode#void	Called only via Reflection API
svc.ecst.universalNodes.VarDecl.VarDecl@String@ECSTNode#void	Called only via Reflection API
svc.ecst.visualise.Visualiser.browseBtnClicked#void	GUI method
svc.sneipl.filter.SoftNet.LinkTransformer.transform@SNLink#Number	Call back method
svc.sneipl.filter.SoftNet.NodeTransformer.transform@SNNode#String	Call back method
svc.sneipl.filter.SoftNet.SNNode.getType#GDNNodeType	Unused
svc.sneipl.FuncCallResolver.dumpCandidates@LinkedList#void	Unused
svc.sneipl.gdn.GDN.LinkTransformer.transform@GDNLink#Number	Call back method

As it can be observed the majority of isolated nodes given in Table I are actually constructors defined in classes that represent different eCST universal nodes. The main characteristic of those classes is that they only define a constructor which invokes the constructor of the super class. At the moment SNEIPL resolves only explicit function calls (calls for which the name of invoked is explicitly stated), while indirect function calls (e.g., through function pointers, variables of procedural data types or via language-specific keywords such as *super* and *this* in the case of Java) are not yet supported. Objects representing universal nodes in eCSTs are instantiated using configurable factory pattern and the Java Reflection API:

- There is a mapping of SNEIPL relevant eCST universal nodes to fully qualified names of SNEIPL classes representing eCST universal nodes.
- The class that loads an eCST relies on the previously mentioned map to determine the name of the class representing currently processed eCST node. The constructor of the class is invoked using the Java Reflection API.

In other words, the calls to constructors which instantiate universal nodes of loaded eCSTs cannot be detected by any static source code analysis method. Generally speaking, function calls via reflection

are hard to detect statically. On the other hand, static analysis tools should be able to detect indirect function calls made via language-specific keywords. In the case of SNEIPL, the problem of indirect function calls via language-specific keywords can be solved by introducing new eCST universal nodes that are specializations of the `FUNCTION_CALL` universal node or even more simply by eCST post-processing which does not change the structure of eCST: each *this/super* token in an eCST can be simply rewritten by the name of class/super-class (those names are marked by `CONCRETE_UNIT_DECL` and `EXTENDS` universal nodes, respectively) in order to make the call explicit by name.

The SNEIPL FUGV network contains 82 isolated nodes (15.81% of the total number of nodes in the network). 4 isolated nodes represent variables, while 78 nodes represent functions. Table II lists isolated nodes that represent class attributes in the SNEIPL FUGV network. As it can be seen only one class attribute defined in SNEIPL is actually unused, while the other three are automatically generated serialization identifiers that are not used by SNEIPL methods.

Table II. Isolated class attributes in the SNEIPL FUGV network.

Attribute name	Explanation
<code>svc.ecst.ECSTTypedNode.typeResolved</code>	Unused
<code>svc.ecst.visualise.Show.serialVersionUID</code>	Serial version UID
<code>svc.ecst.visualise.Visualiser.serialVersionUID</code>	Serial version UID
<code>svc.ecst.ECSTLoaderException.serialVersionUID</code>	Serial version UID

We manually inspected 78 nodes representing methods that are isolated in the FUGV network:

- 20 of them represent methods that are defined in classes which do not define any class attributes. A majority of them are nodes representing constructors of classes that correspond to eCST universal nodes (classes that only define a constructor). Other nodes from this category correspond to method declarations in the only interface contained in SNEIPL (*SymTab*).
- Other 58 methods (17.68% of the total number) are methods that are defined in classes which have class attributes, but do not use them. Those are local (private) methods which only process their arguments, methods which do not use class attributes but invoke other methods which access to class attributes, and simple static utility methods.

5. CONCLUSIONS

In this paper we demonstrated that the source code of a static program analysis tool can be used to validate the tool. More specifically, we applied SNEIPL, extractor of software networks, to its own source code. The analysis of extracted package collaboration network from SNEIPL source code showed that SNEIPL is able to recover its own architecture on the highest level of abstraction. We also performed the analysis of isolated nodes in obtained networks. This analysis revealed unused software entities defined in SNEIPL enabling us to improve its implementation.

REFERENCES

- David F. Bacon and Peter F. Sweeney. 1996. Fast static analysis of C++ virtual function calls. In *Proceedings of the 11th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA '96)*. ACM, New York, NY, USA, 324–341. DOI: <http://dx.doi.org/10.1145/236337.236371>
- David Binkley. 2007. Source Code Analysis: A Road Map. In *2007 Future of Software Engineering (FOSE '07)*. IEEE Computer Society, Washington, DC, USA, 104–119. DOI: <http://dx.doi.org/10.1109/FOSE.2007.27>
- Zoran Budimac, Gordana Rakić, and Miloš Savić. 2012. SSQSA architecture. In *Proceedings of the Fifth Balkan Conference in Informatics (BCI '12)*. ACM, New York, NY, USA, 287–290. DOI: <http://dx.doi.org/10.1145/2371316.2371380>
- Alessandro P. S. de Moura, Ying-Cheng Lai, and Adilson E. Motter. 2003. Signatures of small-world and scale-free properties in large computer programs. *Phys. Rev. E* 68, 1 (Jul 2003), 017102. DOI: <http://dx.doi.org/10.1103/PhysRevE.68.017102>

- Stphane Ducasse, Michele Lanza, and Sander Tichelaar. 2000. MOOSE: an extensible language-independent environment for reengineering object-oriented systems. In *2nd International Symposium On Constructing Software Engineering Tools (COSET 2000)*.
- Jrgen Ebert, Bernt Kullbach, Volker Riediger, and Andreas Winter. 2002. GUPRO: generic understanding of programs – an overview. In *Electronic Notes In Theoretical Computer Science*, Vol. 72. 47–56. DOI: [http://dx.doi.org/10.1016/S1571-0661\(05\)80528-6](http://dx.doi.org/10.1016/S1571-0661(05)80528-6)
- S. Jenkins and S. R. Kirk. 2007. Software architecture graphs as complex networks: a novel partitioning scheme to measure stability and evolution. *Information Sciences* 177 (June 2007), 2587–2601. Issue 12. DOI: <http://dx.doi.org/10.1016/j.ins.2007.01.021>
- Holger M. Kienle and Hausi A. Müller. 2010. Rigi - an environment for software reverse engineering, exploration, visualization, and redocumentation. *Science of Computer Programming* 75, 4 (2010), 247–263. DOI: <http://dx.doi.org/10.1016/j.scico.2009.10.007>
- Panagiotis Louridas, Diomidis Spinellis, and Vasileios Vlachos. 2008. Power laws in software. *ACM Trans. Softw. Eng. Methodol.* 18, 1, Article 2 (Oct. 2008), 26 pages.
- Gail C. Murphy, David Notkin, William G. Griswold, and Erica S. Lan. 1998. An empirical study of static call graph extractors. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 7, 2 (1998), 158–191. DOI: <http://dx.doi.org/10.1145/279310.279314>
- Diego Puppini and Fabrizio Silvestri. 2006. The social network of Java classes. In *Proceedings of the 2006 ACM symposium on Applied computing (SAC '06)*. ACM, New York, NY, USA, 1409–1413. DOI: <http://dx.doi.org/10.1145/1141277.1141605>
- G. Rakić and Z. Budimac. 2011a. Introducing enriched concrete syntax trees. In *Proceedings of the 14th International Multiconference on Information Society (IS), Collaboration, Software And Services In Information Society (CSS)*. 211–214.
- G. Rakić and Z. Budimac. 2011b. SMILE Prototype. In *Proceedings of the International Conference of Numerical Analysis and Applied Mathematics (ICNAAM), Symposium on Computer Languages, Implementations and Tools (SCLIT)*. 544–549. DOI: <http://dx.doi.org/10.1063/1.3636867>
- Gordana Rakić, Zoran Budimac, and Miloš Savić. 2013. Language independent framework for static code analysis. In *Proceedings of the 6th Balkan Conference in Informatics (BCI '13)*. ACM, New York, NY, USA, 236–243. DOI: <http://dx.doi.org/10.1145/2490257.2490273>
- Aoun Raza, Gunther Vogel, and Erhard Plödereder. 2006. Bauhaus: a tool suite for program analysis and reverse engineering. In *Proceedings of the 11th Ada-Europe international conference on Reliable Software Technologies (Ada-Europe'06)*. Springer-Verlag, Berlin, Heidelberg, 71–82. DOI: http://dx.doi.org/10.1007/11767077_6
- Miloš Savić, Mirjana Ivanović, and Miloš Radovanović. 2011. Characteristics of Class Collaboration Networks in Large Java Software Projects. *Information Technology and Control* 40, 1 (2011), 48–58. DOI: <http://dx.doi.org/10.5755/j01.itc.40.1.192>
- Miloš Savić, Miloš Radovanović, and Mirjana Ivanović. 2012a. Community detection and analysis of community evolution in Apache Ant class collaboration networks. In *Balkan Conference in Informatics, 2012, BCI '12, Novi Sad, Serbia, September 16-20, 2012*. 229–234. DOI: <http://dx.doi.org/10.1145/2371316.2371361>
- Miloš Savić, Gordana Rakić, Zoran Budimac, and Mirjana Ivanović. 2012b. Extractor of software networks from enriched concrete syntax trees. *AIP Conference Proceedings* 1479, 1 (2012), 486–489. DOI: <http://dx.doi.org/10.1063/1.4756172>
- Miloš Savić, Gordana Rakić, Zoran Budimac, and Mirjana Ivanović. 2014. A Language-independent Approach to the Extraction of Dependencies Between Source Code Entities. *Inf. Softw. Technol.* 56, 10 (Oct. 2014), 1268–1288. DOI: <http://dx.doi.org/10.1016/j.infsof.2014.04.011>
- Craig Taube-Schock, Robert J. Walker, and Ian H. Witten. 2011. Can We Avoid High Coupling? In *ECOOOP 2011 Object-Oriented Programming*, Mira Mezini (Ed.). Lecture Notes in Computer Science, Vol. 6813. Springer Berlin Heidelberg, 204–228. DOI: http://dx.doi.org/10.1007/978-3-642-22655-7_10
- A. Telea, H. Hoogendorp, O. Ersoy, and D. Reniers. 2009. Extraction and visualization of call dependencies for large C/C++ code bases: A comparative study. In *5th IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT 2009)*. 81–88. DOI: <http://dx.doi.org/10.1109/VISSOFT.2009.5336419>
- S. Valverde and V. Solé. 2007. Hierarchical small worlds in software architecture. *Dyn. Contin. Discret. Impuls. Syst. Ser. B: Appl. Algorithms* 14(S6) (2007), 305–315.
- Lei Wang, Pengzhi Yu, Zheng Wang, Chen Yang, and Qiang Ye. 2013. On the evolution of Linux kernels: a complex network perspective. *Journal of Software: Evolution and Process* 25, 5 (2013), 439–458. DOI: <http://dx.doi.org/10.1002/smr.1550>
- R. Wheeldon and S. Counsell. 2003. Power law distributions in class relationships. In *Proceedings of the Third IEEE International Workshop on Source Code Analysis and Manipulation*. 45–54. DOI: <http://dx.doi.org/10.1109/SCAM.2003.1238030>

Application Challenges of the I/W/SW-OT Paradigm

MUHAMED TURKANOVIĆ and MARKO HÖLBL, University of Maribor, Faculty of Electrical Engineering and Computer Science

The Internet of Things (IOT) paradigm is increasingly infiltrating into our lives. Although still not standardised it already expanded into new paradigms, e.g. Web of Things and Social Web/Internet of Things. Despite the fact that numerous IOT applications already exist, there does not exist any comprehensive methodology or tool for developing IOT applications. Furthermore, the development of such applications is highly challenging because of various IOT particularities as the heterogeneity, resource constrained architecture, scalability, etc. In this paper we evaluate the IOT application development challenges and present some recent methodological solutions.

Categories and Subject Descriptors: D.2.4 [Software Engineering] Coding Tools and Techniques; D.2.11 [Software Architectures] Domain-specific architectures; C.2.1 [Network Architecture and Design] Wireless communication

General Terms: Internet of Things

Additional Key Words and Phrases: Internet of Things, Web of Things, Social Web of Things, Wireless Sensor Network, Applications, Challenges, Development

1. INTRODUCTION

The Internet today is not what it was a decade ago. Social networking was emerging with MySpace and Facebook was an unknown term for almost everyone on the planet. Similarly the iPhone was none existing and a smart phone with the Android OS will come only after few years. Cloud Computing and Software as a Service (SaaS) were concepts known only to specific industry and researchers. Now, Facebook reports about 1.4 billion monthly active users [Facebook, Inc. 2015], owners of an iPhone count about 700 million worldwide [Ingraham 2015], and there is already about 1 exabyte of data stored in the cloud [Woods 2015]. But the interesting difference between today and a decade ago is not hidden in the numbers but in the diversity and heterogeneity of devices connected to the Internet. Specifically, there is a large number of different types of (smart) devices or objects which are interconnected among each other and with the Internet, and thus create a pervasive and ubiquitous network called the Internet of Things (IOT).

As aforementioned, the IOT paradigm exploits the pervasiveness, abundance and versatility of objects which are interconnected. The objects are called *smart objects / things*, since they can sense, compute, communicate and integrate with the environment and with each other. These smart things are also readable, locatable, addressable and controllable via the Internet.

The IOT paradigm is not defined and bound with specific environments, platforms or technologies. It rather presents an upper layer added to the current Internet structure which calls for its exploitation. A different story is with the Web of Things (WOT), which can also be seen as an under-layer of the IOT. The WOT relies on the idea of evolving the Web towards a state, where smart objects are directly

Author's address: M. Turkanović, 2000 Maribor, Slovenia, email: muhamed.turkanovic@gmail.com, tel: +386-40-303-874; M. Hölbl, 2000 Maribor, Slovenia, email: marko.holbl@um.si, tel:+386-2-220-7361.

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

In: Z. Budimac, M. Heričko (eds.): Proceedings of the 4th Workshop of Software Quality, Analysis, Monitoring, Improvement, and Applications (SQAMIA 2015), Maribor, Slovenia, 8.-10.6.2015. Also published online by CEUR Workshop Proceedings (CEUR-WS.org, ISSN 1613-0073)

connected to the Web as we know it today, by applying standard and specific Web Protocols [Levä et al. 2014].

A further extension of the IOT and respectively WOT is the so-called Social Web of Things (SWOT). The SWOT integrates smart objects into the social network and thus facilitates continuous interaction between things and humans [Mashal et al. 2015]. A similar paradigm as SWOT is the Social Internet of Things (SIOT), which attempts to create social consciousness for objects [Atzori et al. 2012]. The main focus of both is the convergence of the IOT and the Social Networks.

Even though all the mentioned paradigms are still being researched and the work for their standardization is still in progress, applications are already being proposed and developed [Atzoria et al. 2010]. However, developing applications for the IOT is challenging because of various reasons. Some of the most challenging are heterogeneity and scalability of objects, addressing and networking issues, resource constrained architecture of devices, mobility of some devices, data management, security and privacy, etc.

In this paper we evaluate the challenges the stakeholder encounter while developing applications for the I/W/SW-OT paradigm. Prior to the evaluation we present a brief review on specific paradigms. In order to better understand the main topic of the paper, we also discuss the IOT architecture. Later on, we discuss some proposed solutions for the IOT application development and present some lateral, non-exhaustive utilities, for aiding the IOT application development and a novel comprehensive model-driven-design approach, which aims to make IOT application development easy.

2. IOT, WOT, SWOT PARADIGM

The IOT paradigm is gaining ground in almost every aspect of our life. Already, the number of connected devices exceeds planet's populations by more than a triple and it is estimated that the number will exceed 7 trillion by 2025 [Sangiovanni-Vincentelli 2011; Evans 2011]. These devices are so-called smart things/items/objects, which are not only equipped with usual wireless communication, memory, microprocessors but furthermore with autonomous proactive behaviour, context awareness, collaborative communication, etc. [Atzoria et al. 2010]. An example of such devices are sensors, actuators, RFIDs, NFCs, smartphones, computers, wearables, etc.

Although there are various definitions of the IOT, there is no clear and unambiguous one, because of different visions about it. This is also in correlation with the diversity of the IOT application domains, whereby the most advanced are those of smart grid, smart home, public safety, independent living, medical and healthcare, logistics, agriculture, industrial processing, etc. [Borgia 2014; Atzoria et al. 2010].

The vision of WOT indicates a more sophisticated layer of networking tightly bind with the Web rather than with the hole concept of the Internet. It leverages technologies like the Internet protocol which is the backbone of the Web. Furthermore the WOT relies much on objects like sensors and smartphones which generate massive amount of data, thus highlighting the importance of the WOT in the era of big data and cloud [Chena et al. 2014]. Therefore the vision of WOT leads to a development of more sophisticated applications which combine any physical interconnected object with the Web Layer. The development of WOT applications has become more flexible with Web-Service-enabled devices, which exploit different existing Web technologies (URI, HTTP, etc.) [Castro et al. 2014; Mashal et al. 2015]. The main contribution of the WOT is the so-called CoAP, a lightweight Web Service based protocol for IOT devices.

A further extensions of the WOT into the Web as we know today (e.g. Web 2.0) is envisioned with the SWOT. The SWOT integrates smart objects into the pervasive social networks (e.g. Facebook) and it facilitates the continuous and automated interaction between physical devices and people [Mashal et al. 2015]. The SWOT enables social network users to leverage their Web-enabled smart objects in

order to access, manage and share object's resources. The paradigm is the logical natural evolution of the WOT.

3. ARCHITECTURE

In order to better understand the challenges of IOT application development we briefly present the IOT architecture and the technologies involved with the paradigm.

Previous classic applications focused on specific dedicated devices with proprietary ICT infrastructure. However IOT applications will share infrastructure, environment and network elements as shown on Figure 1. Figure 1 presents a non-exhaustive list of technologies and protocols, since a vast number of additional various devices and technologies exist.

Authors [Borgia 2014] presented three different phases where the interaction between the cyber (i.e. Internet) and the physical (i.e. Things) world takes place. The first phase is the collection-phase, which is combined of devices (i.e. sensor, actuator, RFID, etc.), responsible for the sensing of physical phenomenon and technologies responsible for the collection of data. The collection is typically done using short range communications (e.g. Bluetooth, IEEE 802.15.4, NFC, etc.). The second phase, called the transmission phase, has the job, to transfer the collected data to the applications over the network. Devices from the first phase are connected to the network via a diverse set of gateways and heterogeneous communication technologies. The most well-known communication technologies used are: IEEE 802.3 (i.e. wired), IEEE 802.11 family (i.e. wireless), IEEE 802.16 (i.e. WiMAX), etc. Lastly, the processing, managing and utilization phase is responsible for forwarding data to the applications and services, processing and analysis of data using semantics, data aggregation, etc. The Service platform & Enabler acts as a abstraction machine, hence hiding the heterogeneity of IOT hardware, software, technologies and protocols.

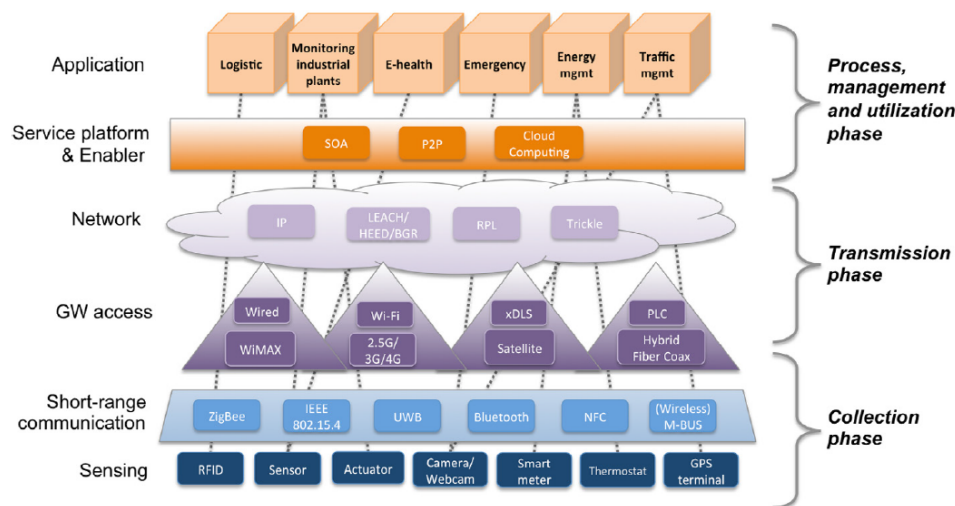


Fig. 1. Representation of some IOT technologies and protocols [Borgia 2014].

4. CHALLENGES

In this section we discuss important IOT specific challenges and specifically, the challenges for developing IOT applications.

Borgia [2014] has summarized the basic IOT challenges into nine groups. The 1st is the **network architecture and system design**. Some essential information about the network architecture was already presented in the previous section. The challenge that reflects on the first group is finding a scalable, flexible and cost efficient architecture for the complex IOT environment. The 2nd group is the **addressing and naming**, which implies the issues of efficient retrieving of all content produced by various heterogeneous IOT end-devices. Classical addressing (i.e. IPv4) is not possible since the addressing space is already exhausted, while the IPv6 is not compatible with the IEEE 802.15.4 communication protocol used by the majority of IOT end-devices (i.e. sensors) [Borgia 2014]. Furthermore there is a problem with the address retrieval. This issue is in combination with the third group presented by Borgia [2014], i.e. **object's mobility**. This is due to the fact that a big part of IOT objects are mobile and thus not fixed (e.g. RFID, sensors, NFC, etc.). The 4th group is **M2M communication**, which encompasses routing problems and end-to-end reliability. The 5th challenge group is the challenge of designing **flexible GW**, which need to be versatile and flexible in order to efficiently manage various nationalities (e.g. access, resources, QoS, etc.). The 6th challenge group are solutions for efficient **device management**, which take into account the heterogeneity of IOT end-devices. The management of devices needs to be enabled remotely. The 7th group concentrates on the **management of data**, precisely management of Big Data, which is the product of a huge amount of IOT devices. Furthermore, managing IOT data is challenging because of the variety of different data properties and semantics. The last two groups are the **traffic characterization** and **security**. The former deals with the problem that the characteristic of the IOT generated traffic is unknown and the later deals with the basic security requirements every application should provide (i.e. confidentiality, integrity, non-repudiation, availability, etc.). Although not specifically specified, one of the biggest challenges is the resource constrained architecture of IOT end-devices (i.e. battery-powered, low processing power, small storage space, etc.) [Akyildiz et al. 2002]. Every aspect of a complete IOT application should take these constraints into consideration since typical solutions may be inappropriate due to the energy requirement. Ensuring such a degree of security is highly challenging due to the IOT characteristics.

Although the basic IOT challenges can easily be transferred to challenges for developing IOT applications, there are still some specific ones as presented by Patel and Cassou [2012]. **Lack of division role** is one of the disclosed challenges, which refers to the multi-disciplined knowledge-process requirement. Thus, for developing an IOT applications, a set of skills are required as domain expertise, deployment-specific knowledge, application design, implementation knowledge and platform-specific knowledge. Another challenge presented by [Patel and Cassou 2012] is the **heterogeneity**, which was already mentioned in the previous paragraph. The application stakeholders have to have in mind IOT heterogeneity in terms of devices, platforms, protocols, etc., since these can largely affect the application design and code. Another challenge is the **scalability**, since IOT end-devices are distributed among a huge amount of different systems and can count in thousands (e.g. WSN) [Akyildiz et al. 2002]. Hence it is very demanding encompassing all the vastness into a practical solution. The last challenge presented by [Patel and Cassou 2012], are **different life cycle phases** (i.e. development, deployment, maintenance). At each mentioned phase, the application logic has to be adapted to a various number of different heterogeneous entities, platforms, devices, etc.

5. SOLUTIONS

In regard to the basic IOT challenges, some solutions are already presented and implemented. A comprehensive list of these solutions can be found in [Borgia 2014; Atzoria et al. 2010].

Alongside to these layer-specific solutions, some lateral, non-exhaustive utilities are also proposed for the IOT application development. An example is the CoAP library proposed by Castro et al. [2014] and designed for the WOT paradigm, i.e. covering the application logic between Web Browsers as Web

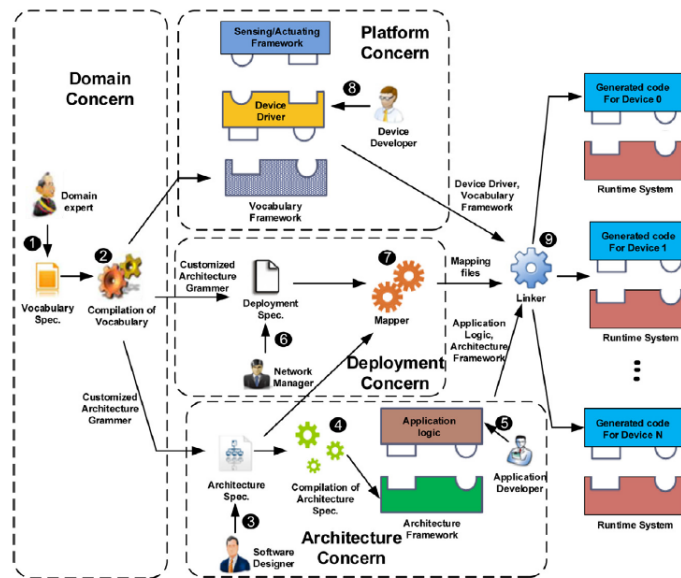


Fig. 2. Conceptual framework for developing IOT applications [Patel and Cassou 2012].

Clients and IOT end-devices. The library is written in Java and it is embedded into a HTML5 Web Site, which in turn is simply accessible through JavaScript. Similar to this approach, there are also others like the Cooper [Matthias 2011] or Actinium [Matthias et al. 2012]. The former is a Firefox and Chrome plugin which supports CoAP browsing. The later is a server-based runtime container for scriptable IOT applications.

Furthermore, research is also done on analysing different approaches for interoperable services (i.e. WS-* and RESTful Web Services) in order to conclude which are better suited for building IOT applications [Guinard et al. 2011].

Unfortunately, there exists no high-level IOT application development tool or methodology, but rather some network specific solutions, which cover only limited subsets of IOT as pervasive computing or sensor networks [Pathak and Prasanna ; Cassou et al. 2012]. Recently, a comprehensive model-driven-design approach, which aims to make IOT application development easy is presented by Patel and Cassou [2012]. The authors have fragmented the IOT application development into several specific aspects, while also integrating a set of high-level languages. They also provide some automation techniques at different phases in order to make the development easier.

The authors have represented the aforementioned aspects or concerns in a conceptual model, which is divided into four concepts, i.e. the domain-specific concept, functionality-specific concept, deployment-specific concept and platform-specific concept [Patel and Cassou 2012]. This conceptual model is later on propagated into a development methodology, which in term result in a conceptual framework for IOT applications development. An overview of the framework is presented in Figure 2.

Among the key features of the methodology presented is the divisibility of application stakeholders into distinct roles according to the four concepts of the conceptual model, i.e. domain expert, software designer, application developer, device developer and network manager. This facilitates the aforementioned challenge called *lack of division role*.

6. CONCLUSION AND FUTURE WORK

In this paper we briefly presented some basic challenges, which stakeholders encounter, while developing IOT-specific application. These challenges need to be addressed comprehensively in order to cover all the possible present and future outcomes. Furthermore, we put forward some recent proposal for aiding the IOT development. Our future objective is to present an exhaustive survey on the state of the art solutions. This way we hope to infer the best practices, methodologies or tools for an easier and robust IOT application development.

REFERENCES

- I. F. Akyildiz, Y. Sankarasubramanian W. Su, and E. Cayirci. 2002. Wireless sensor networks: a survey. *Computer Networks* 38, 4 (2002), 393–422.
- Luigi Atzori, Antonio Iera, Giacomo Morabito, and Michele Nitti. 2012. The Social Internet of Things (SIoT) – When social networks meet the Internet of Things: Concept, architecture and network characterization. *Computer Networks* 56 (2012), 3594–3608.
- Luigi Atzoria, Antonio Ierab, and Giacomo Morabito. 2010. The Internet of Things: A survey. *Computer Networks* 54, 15 (2010), 2787–2805.
- Eleonora Borgia. 2014. The Internet of Things vision: Key features, applications and open issues. *Computer Communications* 54 (2014), 1–31.
- Damien Cassou, Julien Bruneau, Charles Consel, , and Emilie Balland. 2012. Toward a Tool-Based Development Methodology for Pervasive Computing Applications. *IEEE Transactions on Software Engineering* 38, 6 (2012), 1445–1463.
- Miguel Castro, Antonio J. Jara, and Antonio F. Skarmeta. 2014. Enabling end-to-end CoAP-based communications for the Web of Things. *Journal of Network and Computer Applications* In Press (2014).
- Guoqing Chena, Paulo Goesb, Harry Jiannan Wangd, Qiang Weia, and J. Leon Zhaoc. 2014. Guest Editorial: Business applications of Web of Things. *Decision Support Systems* 63 (2014), 1–2.
- Dave Evans. 2011. *The internet of things: How the next evolution of the internet is changing everything*. Technical Report. https://www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf Retrieved 10.7.2014.
- Dominique Guinard, Iulia Ion, and Simon Mayer. 2011. In Search of an Internet of Things Service Architecture: REST or WS-*? A Developers' Perspective (*Proceedings of MobiQuitous*).
- Nathan Ingraham. 2015. Apple has sold 700 million iPhones, 25 million Apple TVs. (March 2015). <http://www.theverge.com/2015/3/9/8164357/apple-watch-event-700-million-iphones-sold> Retrieved 27.4.2015.
- Tapio Levä, Oleksiy Mazhelis, and Henna Suomi. 2014. Comparing the cost-efficiency of CoAP and HTTP in Web of Things applications. *Decision Support Systems* 63 (2014), 23–38.
- Ibrahim Mashal, Osama Alsaryrah, Tein-Yaw Chung, Cheng-Zen Yang, Wen-Hsing Kuo, and Dharma P. Agrawal. 2015. Choices for interaction with things on Internet and underlying issues. *Ad Hoc Networks* 28 (2015), 68–90.
- Kovatsch Matthias. 2011. Demo abstract:human–CoAP interaction with copper (*Proceedings of the 7th IEEE international conference on Distributed Computing in Sensor Systems*).
- Kovatsch Matthias, Lanter Martin, and Duquenooy Simon. 2012. Actinium: a RESTful runtime container for scriptable Internet of Things applications (*Proceedings of the 3rd international conference on the Internet of Things (IoT2012)*). Wuxi, China.
- Pankesh Patel and Damien Cassou. 2012. Enabling high-level application development for the Internet of Things. *The Journal of Systems and Software* 103 (2012), 62–84.
- Animesh Pathak and Viktor K. Prasanna. *High-Level Application Development for Sensor Networks: Data-Driven Approach*. Springer Berlin Heidelberg, 865–891.
- Facebook, Inc. 2015. Facebook Reports First Quarter 2015 Results. (April 2015). <http://investor.fb.com/releasedetail.cfm?ReleaseID=908022> Retrieved 27.4.2015.
- A. Sangiovanni-Vincentelli. 2011. Let's get physical: adding physical dimensions to cyber systems (*Internet of Everything Summit*). Rome.
- Jack Woods. 2015. 20 cloud computing statistics every CIO should know. (January 2015). <http://siliconangle.com/blog/2014/01/27/20-cloud-computing-statistics-tc0114/> Retrieved 27.4.2015.

Influence of Cultural Issues on Data Quality Dimensions

TATJANA WELZER, MARKO HÖLBL, LILI NEMEC ZLATOLAS, MARJAN DRUZOVEC, University of Maribor

Successful software – like information system is meaning nothing if the work is not supported by correct and high quality data, what means, that the data are an important part of software or application especially data applications. Increasing quantity of data and demand for integrated and complex data applications requires high quality in data modelling and data. New concepts, tools and techniques for a database modelling, development and retrieval are required with a final goal: better data and information quality. One of the possible solutions for higher data quality is the integration of cultural aspects. Cultural aspects include different viewpoints, including country dependent parameters and business and domain dependent cultural issues. As a consequence, data quality as well as information quality of applications - software improves if the mentioned approach is applied. In this paper the influence of cultural issues on the data quality dimensions and Deming's Fourteen Points will be presented and discussed.

Categories and Subject Descriptors: **H.2.1 [Database Management]:** Logical Design—*Data models*; **K.6.4 [Management of Computing and Information System]:** System Management—*Quality assurance*;

General Terms: Information Quality

Additional Key Words and Phrases: Data Quality, Data Modelling Quality, Culture, Cultural Issues, Deming's Fourteen Points

1. INTRODUCTION

To improve data quality demands in today increasing quantity of data, various approaches are used. Unfortunately quite often these approaches only vaguely take into consideration that the prerequisite for the high data quality are data quality dimensions. The majority of such approaches do not take into account the influence of cultural issues.

Information technology has automated many operations and made data available to more applications and people. However, the progress of information technology also had an impact on data quality by worsening it. Because users often assume that digital data are correct, the guilt is often put on data and its incorrectness. These problems can grow out of proportions especially in data warehouses and big data environments, as well as on the Internet [Welzer 1998], [Welzer 2013]. The data quality dimension needs to get a new presentation and understanding which includes cultural issues.

In general data quality is multidimensional and complex, and involves not only data management and modelling but also analysis, quality control and assurance, storage and presentation. As stated by Strong et al. [Strong 1997], data quality is related to a specific use case and cannot be assessed independently of a specific domain and / or user. In a database the data does not have actual quality or value [Dalcin 2004] it only has potential value which is harvested when data is used. In addition, English has introduced information (data) quality as data's ability to satisfy customers and to meet customers' needs [English 1999], whereas Redman, suggested that data must be accessible, accurate, timely, complete, consistent with other sources, relevant, comprehensive, provide a proper level of detail, be easy to read and easy to interpret [Redman 2001]. In such a sense a data administrator needs to consider what may need to be done with the data to increase its usability, increase its potential use and relevance, and make it suitable for a wider range of purposes and users [Chapman 2005].

To fulfil Chapman's statement and the before discussed finding we have to give the conceptual view of data much more attentions by incorporating data quality dimensions and cultural issues. The later have to be incorporated and business and domain aspects have to be considered.

In Chapter 2 we will briefly present some notions, including data and information quality, quality dimensions and Deming's Fourteen Points. An overview of cultural issues is described in chapter 3. The main goal of chapter 4 is to introduce the influence of cultural issues on data quality dimensions and

Author's address: T. Welzer, 2000 Maribor, Slovenia, email: tatjana.welzer@um.si, tel: +386-2-220-7299; M. Hölbl, 2000 Maribor, Slovenia, email: marko.holbl@um.si; L. Nemeč Zlatolas, 2000 Maribor, Slovenia, email: llili.nemeczlatolas@um.si; M. Druzovec, 2000 Maribor, Slovenia, email: marjan.druzovec@um.si

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

In: Z. Budimac, M. Heričko (eds.): Proceedings of the 4th Workshop of Software Quality, Analysis, Monitoring, Improvement, and Applications (SQAMIA 2015), Maribor, Slovenia, 8.-10.6.2015. Also published online by CEUR Workshop Proceedings (CEUR-WS.org, ISSN 1613-0073)

Deming's Fourteen Points. Finally we will conclude with a summary of the proposed concepts and future research in chapter 5.

2. DATA QUALITY, INFORMATION QUALITY AND DATA QUALITY DIMENSIONS

The concept of quality is difficult to describe because of its amorphous nature and various definitions presented by different authors. This results in the facts that different authors tend to emphasize different aspects of quality [Fox 1997]. When the quality concept was defined, the emphasis was given on how to achieve quality and how to make it compliant with a standard or a specification. Rapid changes in later years have led to new definitions of quality. One of the most well-known and recognized definitions is the IEEE standard definition [IEEE 1998] in which quality is defined as the totality of features and characteristics of a product or service that bears on its ability to satisfy given users' needs.

For further discussion the most important definitions are those of data quality and information quality, as well as the definition of quality dimensions, which are as well presented by different authors.

T.C. Redman defines the data quality in its broadest sense. In his book *Data Quality for Information Age* [Redman 1996] he implies to data quality definition the relevance of intended uses and sufficient details as well as quality with a high degree of accuracy and completeness, consistent with other sources and presented in appropriate ways.

Giri Kumar Tayi and Donald P. Bollou as guest editors of *Examining Data Quality in the Communications of the ACM*, have defined the term data quality as fitness for use which implies that the concept of data quality is relative [Tayi 1998]. Data appropriate for one use may not possess sufficient quality for another use. Or opposite, already used data comply with some kind of quality. A related problem with multiple users of data is also that of semantics. The data designer and/or initial user may fully agree with same definitions regarding the meaning of the various data items, but probably other users will not share the same view. Such problems are becoming increasingly critical as organizations implement data warehouses, using big data or taking into account different cultural aspects, according to business and expert areas. At the same time the conceptual view on data including cultural issues is more and more important owing to the facts that it can a possible solution for the mentioned problems.

The data quality definition of Ken Orr [Orr 1998] introduces a kind of measurement view on the term. It is defined as a measure of the agreement between the data views presented by an information system and the understanding of the same data by the user. Data administrator wants to ensure that data is accurate enough, timely and consistent for the enterprise to survive and make reasonable decisions. However, the most significant problem of data quality is the facts that it often changes. Data in a database is mostly static, but in the real world it is rapidly changing. One reason more to apply a conceptual view influenced by cultural issues.

If defining and understanding data and data quality is difficult and it varies, then defining and understanding information is a hornet's nest. In some environments the term information mistakenly refers to both data and information [Strong 1997]. Data usually refers to information at their early stages of processing and information to the product at a later stage when the meaning is added. Rather than switching between the terms information is used to refer to data or information values at any point in the process. But still we must bear in mind that different information definitions depend on different points of view. For example:

- From the information management point of view, information is simply processed data [Redman 1996].
- From the information theory point of view, information is the non-redundant part of a message [Redman 1996].
- From the information technology for management point of view, information is data that has been organized in a way that is has meaning to the user [Turban 1996]

However once a point of view is fixed, no conflict should arise. Once again it is important to emphasize that the prerequisite for information quality is data quality.

But to get a better view on data quality, particularly from the conceptual point of view, data quality dimensions have to be introduced. Redman defined for the users' perspectives 15 characteristics of an ideal view [Redman 1996]:

- Relevance – data that is needed by the application;
- Obtainability – values should be easily obtained;
- Clarity of definition – all terms should be clearly defined;
- Comprehensiveness – all required data should be available and included;
- Essentialness – unneeded data should not be included;
- Attribute granularity – right level of definitions and abstractions for data;
- Domain precision – appropriates of domains;
- Naturalness – existence in the real world;
- Occurrence identifiability – identification of entities (data);
- Homogeneity – minimization of unnecessary attributes;
- Minimum redundancy;
- Semantic consistency – clear and consistent view;
- Structural consistency;
- Robustness – wide view;
- Flexibility – easy to change.

We also have to emphasize Deming's Fourteen Points. Deming defined his 14 points or key principles with the intention to make easier implementation of changes in companies, departments and teams. They are a guide to the importance of building users awareness. And from that point of view it is important to introduce cultural issues into data quality dimensions.

3. CULTURAL ISUESS

One of the most familiar words in any community is culture. The word itself is used in different combinations and meanings, which leads to many definitions of culture. Culture can be presented as an artistic activity, as a social, philological or anthropological concept or as a culture of groups, societies and countries. The world itself has grown over the centuries to reach its currently broad understanding [Baldwin 2004]. However, culture is not something that we simply absorb; it is something that we have to learn although the common knowledge is mostly opposite.

The area of culture has been studied by well-known researchers and we are faced with different definitions and explanations that are showing us authors' point of view on the topic [Welzer 2011]. Hofstede for example defined culture as a collective phenomenon, because it is shared with people who live or lived within the same social environment. According to Hofstede culture consists of unwritten rules of social game. It is the collective programming of the mind that distinguishes the member of one group or category of people from others [Hofstede 2001]. Lewis, another important researcher, explains that the culture is an integrated pattern of human knowledge, a core belief, and a behaviour that depends upon the capacity for symbolic thought and social learning [Lewis 2007]. Culture also refers to the cumulative deposit of knowledge, experience, beliefs, values, meanings, hierarchies, religion, notions of time, roles, spatial relations, concepts of the universe and material objects and possessions acquired by a group of people in the course of a generation through individual and group striving [Schneider 2003]. If people adjust to cultural differences, they can better face challenges and become better in their own profession [Welzer 2010].

The summary of all these different, but also similar definitions, is the definition given by Rossinski, which understand culture in the frame of a group as a set of unique characteristics that distinguishes its members from another group [Rossinski 2003]. This definition can be easily applied to nations and subgroups, as well as business environments (business and corporate culture) [Welzer 2010].

In addition to other important terms, there is also a very important term: cultural awareness. None or poor cultural awareness means a poor understanding of cross-cultural dialogue, which can lead to blunders and damaging consequences, especially in business, management and advertising, where cultural awareness seems to be of key importance for success. However, engineering, medicine and many other areas are also not immune to it [Hofstede 2004]. According to the definition, cultural awareness is the foundation of communication, and it involves the ability of observing our cultural values, beliefs and perceptions from the outside [Hofstede 2004]. Cultural awareness is important in communication with people from other cultures, and we have to understand that people from different cultural societies might see, interpret and evaluate things in different ways.

A good illustration of culture and cultural awareness can also be found in cultural levels, as defined by Alvesson and Berg [Alverson 1992]. They introduced different levels within the concept of culture [Alverson 1992]: culture in societies and nations, regional and local, business cultures, organizational and corporate, functional subcultures at the organizational level, social groups in the organization, professional and functional cultures. The numbered levels contain subgroups, or rather, specific cultures according to social life, geographical location and business domain, including enterprise and organizational culture [Welzer 2011]. Such a definition of culture is probably more comprehensible to engineers and other business and technical groups because they are more familiar with this presentation of culture.

4. CULTURAL ISSUES AND DATA QUALITY DIMENSIONS

Quality and culture on their own represent two big areas of research and bear high importance in development of new products as well as in behaviour of users. In this paper we would like to introduce the influence of cultural issues on data quality dimensions. In Chapter 2, 15 characteristics and Deming's Fourteen Points adapted for data are presented and are an important introduction to this chapter.

According to 15 characteristics as well as fourteen points an introduction the possible cultural issue was done:

- Relevance – data needed by the application – *different cultural point of view can provide different data needed*
- Obtainability – values should be easily obtained – *according to cultures and flowing laws, the obtaining of some data could be not as easy as expected*
- Clarity of definition – all terms should be clearly defined – *we can find different definitions for same or similar data in different environments*
- Comprehensiveness – all data needed should be available and included – *some cultures may not allow the collection of some data*
- Attribute granularity – right level of definitions and abstractions for data – *it must be clear what are sensitive data for which cultures*
- Domain precision – appropriates of domains – *culture sensitives of domains*
- Naturalness – existing in real world – *counterpart in the real word can differentiate from culture to culture*
- Occurrence identifiability – identification of entities (data) – *different points of view in culture*
- Homogeneity – minimization of unnecessary attributes – *some of the attributes can be results of cultural differences*
- Semantic consistency – clear and consistent view – *strong cultural influence*
- Robustness – wide view – *generalization in the cultural point of view*
- Flexibility – easy change – *limitations from cultural point of view are possible.*

In addition, points of view of influence of cultural issues on data quality dimensions is given by Deming's Fourteen Points for quality management, adapted for data [Redman 1996]. Through this the new philosophy of understanding quality from cultural point of view is becoming even stronger [Redman 1996], [Welzer 1998], [Welzer 2013]:

- Point 1 – Recognize the importance of data and information to the enterprise. *Same data has different meanings and importance in different enterprises and so also cultures.*
- Point 2 – Adopt new philosophy. The enterprise can no longer live with currently accepted levels of data quality. *Introducing cultural issues is confirming this point.*
- Point 3 – Cease dependence on error detection. Eliminate the need for error detection by building accuracy and other quality attributes into processes that create data. *Culturally influenced detection of errors.*
- Point 5 – Constantly improving the systems by which data are produced and used to create value for customers, the enterprise and its stakeholders. *Introducing the cultural issues is an improvement.*
- Point 6 – Institute job training. *Cultural awareness help individuals and organizations to understand how the cultural issues impact data.*

- Point 7 – Teach and institute leadership for supervisors on workers, who produce data. Managers of organizations that produce data must become responsible for quality, not simply numeric production. The entire enterprise productivity will improve with improved data. *Cultural point of view has to be introduced in those activities to confirm improvement for the entire enterprise.*
- Point 9 – Break down barriers between organizations. *Application, functional and business domain ensure a free flow of cultural awareness across the organizational boundaries.*
- Point 12 – Remove barriers standing between data products and their rights to pride in their work. *Cross cultural awareness motivates designers to different solutions and models.*
- Point 13 – Institute training on data and information, their roles in the enterprise and how they may be improved. *The training has to be supported by cultural issues and cultural awareness.*
- Point 14 – Create a structure in top management that recognizes the importance of data and information and their relationships to the rest of the business. *Cultural issues supports this recognition and the top management can always find a support for understanding data in existing models and applications.*

5. CONCLUSION

There is no doubt that data quality is needed, but as with many other activities we have to agree on adequate measures and this is highly demanding activity.

In the case of data quality we have the support different guidelines like Data Quality Dimension Characteristics or Deming's Fourteen Points. Deming's Fourteen Points for Quality Management were adapted for data by Redman [Redman 1996] and in this paper we have suggested an adaptation of Deming's Fourteen Points as well as 15 characteristic, taking cultural issues into consideration. In this paper we have presented characteristics and points that could be adapted for cultural issues. Additionally, we presented cultural issues and comments concerning them.

Cultural issues as well as comments are based on experiences and previous research [Welzer 2010], [Welzer 2011], [Welzer 2013] about data (same or similar data) in different environments and cultures. For example some cultures (business environments) are using maiden name (a possible attribute), while some others do not use it at all and operate only with the family name (an attribute as well) from which the maiden name can be derived, but only in some environments and cultures, what do not guarantee an easy obtainability, neither clear definitions and availability. Also attribute granularity, domain precision and naturalness as well as semantic consistency and robustness are effected in this case. To obtain the cultural issues in numbered characteristic as guidelines the Deming's Fourteen Points adapted for data can be used.

In further research policy and strategy has to be involved to make the guidelines clear and suitable for testing in daily processes connected to data and quality.

REFERENCES

- Alvesson, M., Berg, P.-O. 1992. Corporate Culture and Organizational Symbolism
- Baldwin, E. et al. 2004. Introducing Cultural Studies. Prentice Hall, Harlow. W de Gruyter, New York.
- Chapman, A. D. 2005. Principles of Data Quality, version 1.0. Report for the Global Biodiversity Information Facility, Copenhagen.
- Dalcin, E.C. 2004. Data Quality Concepts and Techniques Applied to Taxonomic Databases. Thesis for the degree of Doctor of Philosophy, School of Biological Sciences, Faculty of Medicine, Health and Life Sciences, University of Southampton. November 2004. 266 pp. http://www.dalcin.org/eduardo/downloads/edalcin_thesis_submission.pdf [Accessed June 1, 2015].
- English, L.P. 1999. Improving Data Warehouse and Business Information Quality: Methods for Reducing Costs and Increasing Profits. New York: John Wiley & Sons, Inc.
- Fox, C. and Frakes, W. 1997. Elements of the Quality Paradigm. Communications of the ACM, vol. 40, no. 6, pp. 26. IEEE (1989).
- Hofstede, G. 2001. Culture's Consequences, Comparing Values, Behaviors, Institutions and Organizations Across Nations, Sage Publications, Thousand Oaks.
- Hofstede, G., Hofstede, G.J. 2004. Cultures and Organizations: Software of the Mind: Intercultural Cooperation and its Importance for Survival, McGraw-Hill, New York.
- IEEE Standards. 1998 – Third Edition. New York: IEEE.
- Lewis, R.D. 2007. When Cultures Collide, Managing Successfully Across Cultures, Nicholas Brealey Publishing, London.
- Orr, K. 1998. Data Quality and Systems Theory. Communications of the ACM, vol. 41, no. 2, 66-71.

- Redman, T.C. 1996. *Data Quality for the Information Age*. Boston, London: Artech House.
- Redman, T.C. 2001. *Data Quality: The Field Guide*. Boston, MA: Digital Press.
- Rosinski, P. 2003. *Coaching Across Cultures*, Nicholas Brealey Publishing, London.
- Schneider, S.C., Barsoux J-L. 2003. *Managing Across Cultures*, Prentice Hall, Harlow
- Strong, D.M., Lee, Y.W. and Wang, R.Y. 1997. 10 Potholes in the Road to Information Quality. *IEEE Computer*, vol. 30, no. 10, 38-46.
- Strong, D.M., Lee, Y.W. and Wang, R.W. 1997. Data quality in context. *Communications of ACM*, vol. 40, no. 5, 103-110.
- Tayi, G.K. and Ballou, W. 1998. Examining Data Quality. *Communications of the ACM*, vol. 41, no. 2, 54-57.
- Turban, E.; MvLean, E. and Wetherbe, J. 1996. *Information Technology for Management*, New York: John Wiley & Sons.
- Welzer, T. and Rozman, I. 1998. *Information Quality by Meta Model*. *Proceedings Software Quality Management VI*. London: Springer, 81-88.
- Welzer, T., Družovec M., Cafnik, P., Zorič Venuti M., Jaakkola H. 2010. Awareness of Culture in e-learning. *ITHET 2010, IEEE*, pp. 312-315.
- Welzer, T., Holbl, M., Družovec, M., Brumen, B., 2011 *Cultural awareness in social media*. *DETECT 2011, New York: ACM, 2011*, pp. 1-5.
- Welzer, T., Družovec, M., Holbl, M., 2013 *Cultural issues as a components of data modelling quality : lecture*, presented on *SQAMIA 2013, 2nd Workshop on software quality, analysis, monitoring, improvement and applications*, Novi Sad, September 2013.