Andreas Fuhr, Wilhelm Hasselbring, Volker Riediger (Eds.) and
Magiel Bruntink, Kostas Kontogiannis (Eds.)

Joint Proceedings of the

First International Workshop on Model-Driven Software Migration (MDSM 2011)

and the

Fifth International Workshop on Software Quality and Maintainability (SQM 2011)

March 1, 2011 in Oldenburg, Germany

Satellite Events of the

15th European Conference on
Software Maintenance and Reengineering
(CSMR 2011)

March 1-4, 2011

Editors' addresses:

Andreas Fuhr, Volker Riediger

University of Koblenz-Landau
Institute for Software Technology
Universitätsstr. 1, 56070 Koblenz, Germany
{afuhr | riediger}@uni-koblenz.de

Wilhelm Hasselbring

University of Kiel
Workgroup Software Engineering
Christian-Albrechts-Platz 4, 24118 Kiel, Germany
wha@informatik.uni-kiel.de

Magiel Bruntink

Software Improvement Group
Amstelplein 1, 1070 NC Amsterdam, The Netherlands
m.bruntink@sig.eu

Kostas Kontogiannis

National Technical University of Athens
School of Electrical and Computer Engineering
Greece
kkontog@softlab.ece.ntua.gr

Published on CEUR Workshop Proceedings,
ISSN: 1613-0073,
Vol. 708 online at http://ceur-ws.org/Vol-708

Contents

Proceedings of MDSM 2011

Proceedings of the

First International Workshop on Model-Driven Software Migration (MDSM 2011)

First International Workshop on Model-Driven Software Migration (MDSM 2011)

March 1, 2011 in Oldenburg, Germany

Satellite Event of IEEE CSMR 2011
March 1-4, 2011

Wilhelm Hasselbring
University of Kiel
Software Engineering Group
Kiel, Germany
wha@informatik.uni-kiel.de

Andreas Fuhr, Volker Riediger
University of Koblenz-Landau
Institute for Software Technology
Koblenz, Germany
afuhr|riediger@uni-koblenz.de

Preface

Welcome to the First International Workshop on Model-Driven Software Migration (MDSM 2011), March 1, 2011 in Oldenburg, Germany.

Model-driven software development (MDSD) and software migration are two different approaches that had been under research separately. In recent years, researches found interesting analogies between both fields.

In software engineering, one of the key principles is abstraction, that is, focusing only on the important aspects while fading-out details. Model-driven software development (MDSD) aims at modeling these important aspects at different levels of abstraction. This allows to design software starting with the "big picture" (abstract level) and approach more concrete levels by adding details to the models until the system is implemented (concrete level).

Software migration aims at converting an old system (legacy system) into a new technology without changing functionality. This implies understanding, how the legacy system is working. For this purpose, legacy code must be leveraged into a higher level of abstraction in order to focus only on the important aspects.

At this point, model-driven software development and software migration meet. Migration projects can benefit from the vision of MDSD by abstracting legacy systems (reverse engineering), transform them and implement the migrated system (forward engineering).

However, both fields of research are not yet entirely understood. Neither is the combination of both fields examined very well.

MDSM 2011

The MDSM workshop brought together researchers and practitioners in the area of model-driven approaches supporting software migration to present and discuss state-of-the-art techniques as well as real-world experiences to stimulate further model-driven migration research activities.

The scope of the MDSM workshop included, but was not restricted to, the following topics:

- Modeling languages, query languages and transformation languages
- Domain Specific Languages for software migration
- Model-integration in repositories
- Model-driven architecture reconstruction or migration
- Model-driven code migration
- Software migration by transforming legacy code
- Model-driven software renovation
- Tools and methods for model-driven migration
- Design patterns for model-driven software migration
- Experience reports

The MDSM workshop was held during the CSMR 2011 main conference on March 1, 2011. The full-day workshop consisted of three thematically grouped sessions:

- one 90 minutes project session, presenting latest research projects in the field of model-driven software migration
- two 90 minutes paper sessions containing paper presentations with plenty of time for discussions.

The proceedings contain the papers and project presentations presented at MDSM 2011. For regular papers, we received eight submissions, from which we accepted four papers based on a rigorous reviewing process. Each paper was reviewed by four program committee members. In addition, we invited three projects in the field of model-driven software migration to present their work and to submit a 2 pages summary of their project.

Organizers

Workshop Chairs

- Wilhelm Hasselbring, Christian-Albrechts-Universität zu Kiel, Germany
- Andreas Fuhr, Universität Koblenz-Landau, Germany
- Volker Riediger, Universität Koblenz-Landau, Germany

Program Committee

- Andy Schürr, Technische Universität Darmstadt, Germany
- Anthony Cleve, Institut National de Recherche en Informatique et en Automatique (INRIA) Lille, France
- Bernhard Rumpe, Rheinisch-Westfälische Technische Hochschule (RWTH) Aachen, Germany
- Dragang Gasevic, Athabasca University, Canada
- Eleni Stroulia, University of Alberta, Canada
- Filippo Ricca, Università degli Studi di Genova, Italy
- Harry Sneed, Central European University Budapest, Hungary & Universität Passau, Universität Regensburg, Germany
- Heinz Züllighoven, Universität Hamburg, Germany
- Jaques Klein, Université du Luxembourg
- Jorge Ressia, Universität Bern, Switzerland
- Jürgen Ebert, Universität Koblenz-Landau, Germany
- Klaus Krogmann, Karlsruhe Institute of Technology, Germany
- Rainer Gimnich, IBM Frankfurt, Germany
- Rocco Oliveto, Università degli Studi di Salerno, Italy
- Romain Robbes, Universidad de Chile, Chile
- Steffen Becker, Universität Paderborn, Germany
- Tudor Girba, Universität Bern, Switzerland

Acknowledgments

The organizers would like to thank all who contributed to the workshop. We thank the authors for their submissions and we especially thank the Program Committee and their sub-reviewers for their good work in carefully reviewing and collaboratively discussing the submissions.

March, 2011

Andreas Fuhr
Wilhelm Hasselbring
Volker Riediger

Model-Driven Migration of Scientific Legacy Systems to Service-Oriented Architectures

Jon Oldevik, Gøran K. Olsen
SINTEF Information and Communication Technology
Forskningsvn 1, 0373 Oslo, Norway
jon.oldevik | goran.olsen at sintef.no

Ute Brönner, Nils Rune Bodsberg
SINTEF Materials and Chemistry
Brattørkaia 17c, 7465 Trondheim, Norway
ute.broenner | nilsrune.bodsberg at sintef.no

***Abstract*—We propose a model-driven and generative approach to specify and generate web services for migrating scientific legacy systems to service-oriented platforms. From a model specification of the system migration, we use code generation to generate web services and automate the legacy integration. We use a case study from an existing oil spill analysis application developed in Fortran and C++ to show the feasibility of the approach.**

***Keywords*-Model-driven engineering, legacy migration, web services**

I. Introduction

A large number of existing systems, especially within data and computationally intensive domains, are based on implementations that are becoming increasingly difficult to maintain and evolve [1], typically in languages like Cobol and Fortran. Competent personnel with knowledge of these technologies is also becoming a scarce resource. Modernisation toward a service-oriented architecture may also open for new business opportunities.

In this paper, we investigate a model-driven approach for migrating legacy systems to service-oriented architectures. Our migration strategy is wrapping of existing legacy components. We use the Unified Modelling Language (UML) to specify migration models, or wrappers, that are fed to model-driven code generators to generate a deployable service. This work has been done in the SiSaS project[1], which has an overall focus of methods and tools for migrating *scientific software* to service-oriented architectures.

We define a migration profile in UML that contains concepts for integrating with existing legacy, such as native libraries, executable programs, and databases, as well as for integrating with existing web services. We establish a modelling approach – a method – for how to specify services using the migration concepts, as well as concepts from SoaML [2]. Our modelling comprises the interfaces and structure of a service, as well as the behaviour of different service parts.

Our goal is to create effective and usable means for migrating legacy systems to service-oriented platforms. Our conjecture is that model-driven and generative techniques can provide these means.

[1]SINTEF Software as a Service

II. Motivating Case Study: Oildrift Simulation

In SINTEF Materials and Chemistry, they have a commercial legacy product for simulating oil drift, which can help predicting the spreading of oil in case of an accidental spill. The system is implemented by a Fortran simulation back-end and a C++ front-end. Now, they want a transition to a service-oriented paradigm to more easily adapt to new customer needs and more flexible business models. Figure 1 illustrates the existing application.

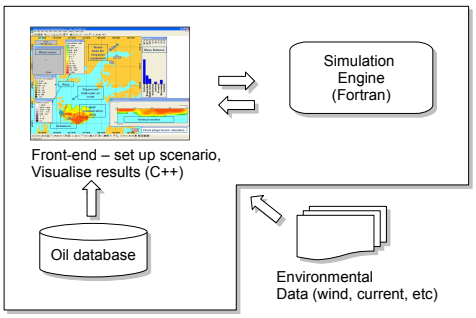


Figure 1. Oildrift Prediction – Legacy Application

The Fortran simulation core is responsible for simulating oil drift based on numerical models. It is invoked from a presentation layer written in C++. All input is file based, and simulation runs in batch mode from some minutes to several days. This approach has worked fine for many years, but there are some apparent challenges with respect to interoperability, integration, and scalability. The goal is to migrate the application to meet new market needs while coping with these challenges.

III. Our approach

We use model-driven engineering techniques to develop the oil drift prediction as a service that wraps the existing simulation engine. UML models are used to specify the service interfaces and the details of the wrapper architecture. From these models, we generate

XML schemas for the web service, Java interface and class implementations of the web service, the architecture of the wrapper, and its behavioural implementation. Wrapping of the C++ front-end is out of our scope, since this will be re-designed to fit a web-based interaction paradigm.

We define a UML *migration profile* to represent semantics of different types of migration features, such as *executables*, *databases*, and *native libraries*. The code generators use this semantics to generate the necessary integration code. Figure 2 illustrates the high-level approach.

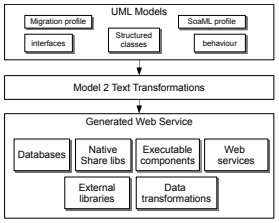


Figure 2. Approach Overview

We use UML interfaces and classes to model the structural parts of the system. Service interfaces define the behaviour, and classes define the internal structure of the services. UML composite structures are used for specifying the service de-composition into parts. The service is decomposed by legacy component parts, which is orchestrated by the service to provide its operations. The behaviour of the service and its contained legacy wrapper components is defined by UML activity diagrams.

To relate the migration models to the service-oriented modelling domain, we use some SoaML concepts to describe services: the stereotype «serviceInterface» is used to denote a service, i.e. the service that wraps the legacy systems. The stereotype «MessageType» is used to specify the data types passed as message input and output of the service.

A. The migration profile

The migration profile contains a set of stereotypes used for adding migration semantics to the UML models. The main purpose of the migration model is to integrate existing legacy functionalities and expose them through well-defined interfaces. To this end, we use standard UML models extended with migration semantics from a profile.

The Component Types: The component types represent different sorts of legacy components that take part in fulfilling the responsibilities of the legacy system. This might be existing shared libraries, executables, Java libraries, databases, or web services. Figure 3 specifies the set of component types that are in the current profile.

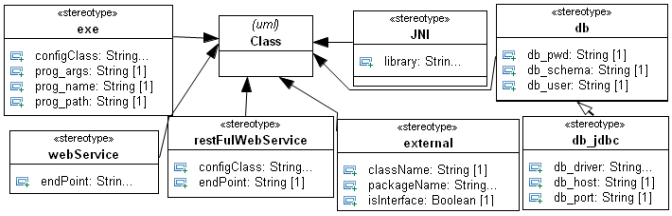


Figure 3. Component types

The stereotype «WebService» denotes the wrapping of an external web service, i.e. a web service client. «RestfulWebService» denotes the wrapping of a restful web service. Its endpoint is an URI that acts as a data source, which is fetched by the wrapper and used locally. «exe» denotes the wrapping of an executable program. «JNI» denotes the wrapping of a native library, such as a windows shared DLL, using Java Native Interface (JNI). «external» denotes integration with external Java libraries, e.g. provided by a *jar* file. «db_jdbc» denotes the wrapping of a JDBC database. Operations defined in classes of this type represent database SQL queries.

The profile additionally provides stereotypes for specific types of operations, such as «asynch» for *asynchronous* operations and «RSOp» for restful service operations. Exceptions can be specified explicitly by classes stereotyped «exception». Throwing and catching of exceptions are specified by dependencies stereotyped «throws» and «catches».

Behaviour – Activities and Actions: Behaviour is declared by operations in components. The behaviour of these operations are defined by associated *activity diagrams*. An activity diagram defines behaviour by sequences and branches of actions that are mapped to statements in code generation. Standard (opaque) actions contain embedded Java code. *CallOperationActions* are used for defining invocations to defined operations of related components. In addition, we define a set of stereotypes in the migration profile for simplifying the action specification:

«return» is used to denote a return from the method execution with a specific value; «assign» is used to denote an assignment of a value to a variable; «setState» is used to denote the setting of an internal state variable, specifically used for asynchronous and long running operations; «setReturn» is used to set the return value of an asynchronous and long running operation; «param» is used to define an input parameter to a *CallOperationAction*. It references a previously defined variable;

finally, «valueparam» is used to define a literal value as an input parameter to a *CallOperationAction*.

B. Modelling the Oildrift Prediction Case

In this section, we exemplify the use of the migration profile on the oildrift prediction case in terms of structure and behaviour.

Service Structure and Interface: The service itself is defined by a SoaML «serviceInterface» class, which implements the service interface with a set of exposed operations (Figure 4). The most interesting of these is the «asynch» operation *predictOilDriftAsynch*, which provides the main service in the oildrift prediction case. Since the execution of a simulation may run for hours, or even days, the operation is declared as asynchronous. The operation will return immediately with only a *session id* to identify the session.

Two additional helper operations are provided for checking execution status (*getStatus*) and to retrieve the result upon termination (*getPredictOilDriftResult*).

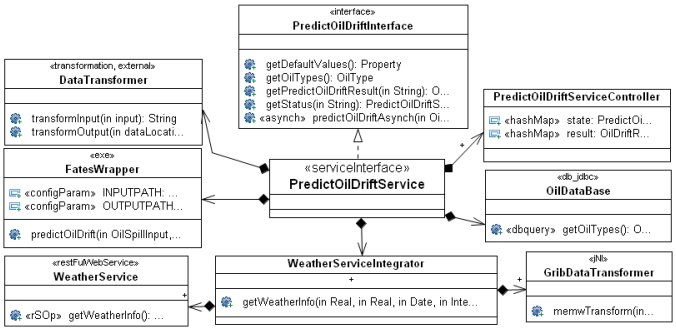


Figure 4. Predict Oildrift – Service and Wrapper Components

The *PredictOilDriftService* is a structured class that contains a set of parts: the *PredictOilDriftServiceController* is the internal orchestration component for the service. All incoming calls are delegated to the controller, which implements the operations of the service. The *DataTransformer* provides operations for transforming input required by the *Fortran* simulation engine, and transforming result data after simulation. The *FatesWrapper* is an «exe» component, which wraps the execution of the *Fortran simulation* program. The *WeatherServiceIntegrator* provides operations for integrating with an external weather data provider. It is further de-composed by two parts: a «restfulWebService» called *WeatherService* and a «JNI»-component called *GribDataTransformer*. The *getWeatherInfo* operation gets weather data from a restful web service that provides binary data in the *GRIB*[2] format. To transform the GRIB-data to the input format used by the simulation engine, an external native library (in this case DLL) is integrated by the *GribDataTransformer*. The *OilDatabase* is a «db_jdbc»-component, which provides oil type information from an SQL database.

The *data types* passed in the service interface are modelled as classes stereotyped using the SoaML stereotype «MessageType». Apart from the stereotype, the data types are specified with standard UML classes with attributes and associations.

Behaviour: Component behaviour is specified for different operations in the migration model. Behaviour is defined by *Activity diagrams* that are associated with the operations they implement.

Figure 5 shows the behaviour of the *predictOilDriftAsynch* operation, which is contained in the *PredictOilDriftServiceController*. All invocations to the service *PredictOilDriftService* is by convention delegated to its *controller* part.

The example with the asynchronous operation is interesting, since it also requires handling of the long-running external executable. In this case, this is handled by providing a second activity diagram, which specifies the behaviour upon termination of the executable.

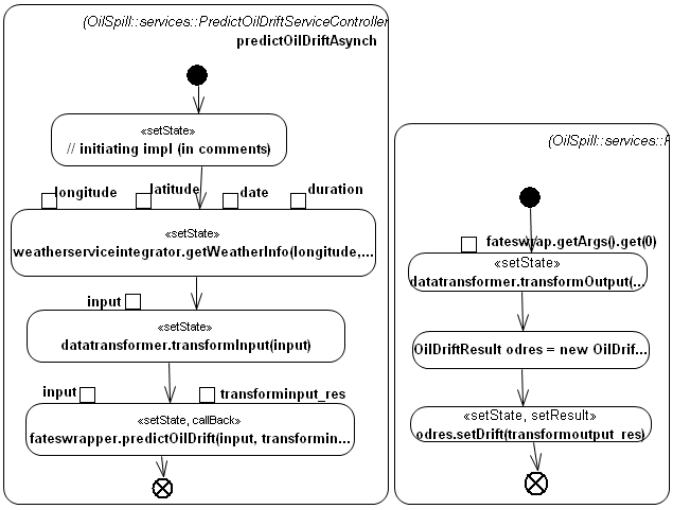


Figure 5. Predict Oildrift Service Behaviour

The first activity diagram specifies the initial behaviour of the operation, up until the call to the *executable* component.

The second activity diagram from Figure 5 specifies the behaviour occurring when the execution of the external program has terminated. When the final action is executed, the service state (for this particular session id) is set to a *ready* state, and the client can fetch the *result* of the service.

Our approach for behavioural modelling is tied to the target language, in this case Java, since we in some cases embed small portions of code inside actions. We

[2]GRIdded Binary, http://www.wmo.int

could get around this by incorporating a richer set of UML actions that can cope with variable declarations, property references, and object creation. This extension will be investigated as part of future work.

C. Code generation

The purpose of the migration models is to automate as much as possible the legacy system migration process. We have developed a set of transformations, or code generators, to support the transition from models to deployable services. They were implemented with the model to text transformation tool MOFScript[3].

IV. Related Work

Dorda et al.[4] give a survey of legacy system modernisation approaches. They distinguish between two main types of modernisation: *white-box* and *black-box* modernisation. White box modernisation requires an understanding of the internal parts of a system, and involves re-structuring, re-architecting, and re-implementing the system. Black-box modernisation is only concerned with the input/output, i.e. the interfaces, of the legacy system, and is often based on wrapping. Our approach can be seen as a model-driven black-box modernisation technique. However, the migration also has flavours of white-box migration to it, in particular in understanding and transforming the legacy data formats.

Within the Object Management Group (OMG), the Architecture-Driven Modernization (ADM) task force [5] is working on standards to support legacy modernisation, such as meta-models for knowledge discovery, software visualisation, and refactoring.

Razavian and Lago [6] present a SOA migration framework – (SOA-MF) – wherein they establish an overall process framework for legacy migration, focusing on recovery and re-engineering, and put it in the context of migration methods such as SMART [7]. Although our work has not focused on re-factoring or re-engineering, the processes targeting legacy discovery and transformation to new architecture has also been addressed in our work.

Canfora et al.[8] present an approach for migrating interactive legacy systems to web services based on wrapping. They define a model (a state machine) of the user interactions, which is the basis for integration with legacy systems through *terminal emulation*. This process is then exposed as a web service.

V. Conclusion and Outlook

We have presented a model-driven approach for legacy migration to service-oriented architectures, where the focus is black-box migration by wrapping legacy components using model-driven and generative techniques. We have defined a UML profile for migration and a set of code generators for generating services and wrapper components. We have tested the migration approach on an oildrift prediction system, by modelling and generating the services and wrappers required for integration with the various legacy components.

At this time, we have not addressed *automation of legacy data mappings*, which is a major concern in legacy modernisation. In the current case study, data mappings between binary data formats where written manually. In future work we will investigate appropriate techniques and tools for specifying data transformations at the model level, and for mapping these to the implementation level.

Acknowledgements

The results reported in this paper are from the SiSaS project *(SINTEF Software as a Service)*. SiSaS is an internal project within SINTEF that focuses on migration of scientific legacy scientific software to services.

References

[1] F. Zoufaly, "Issues and challenges facing legacy systems," *Project Management, developer.com*, 2002.

[2] Object Management Group (OMG), "Service Oriented Architecture Modeling Language (SoaML), FTF Beta 2," OMG, Standard ptc/2009-12-09, 2009.

[3] J. Oldevik, T. Neple, R. Grønmo, J. Aagedal, and A. Berre, "Toward Standardised Model to Text Transformations," in *European Conference on Model Driven Architecture - Foundations and Applications (ECMDA)*. Nuremberg: Springer, 2005, pp. 239–253.

[4] S. Comella-Dorda, K. Wallnau, R. C. Seacord, and J. Robert, "A survey of legacy system modernization approaches," 2000. [Online]. Available: http://handle.dtic.mil/100.2/ADA377453

[5] Object Management Group (OMG), "ADM White Paper: Transforming the Enterprise," OMG, White paper http://www.omg.org/docs/admtf/07-12-01.pdf, 2008.

[6] M. Razavian and P. Lago, "Understanding SOA migration using a conceptual framework," *Czech Society of Systems Integration*, 2010.

[7] S. Balasubramaniam, G. A. Lewis, E. J. Morris, S. Simanta, and D. B. Smith, "SMART: Application of a method for migration of legacy systems to SOA environments," in *Service-Oriented Computing - ICSOC 2008, 6th International Conference, Sydney, Australia, December 1-5, 2008. Proceedings*, ser. Lecture Notes in Computer Science, A. Bouguettaya, I. Krüger, and T. Margaria, Eds., vol. 5364, 2008, pp. 678–690.

[8] G. Canfora, A. R. Fasolino, G. Frattolillo, and P. Tramontana, "Migrating interactive legacy systems to web services," in *CSMR*. IEEE Computer Society, 2006, pp. 24–36.

Model-driven Modernisation of Java Programs with JaMoPP

Florian Heidenreich, Jendrik Johannes, Jan Reimann, Mirko Seifert,
Christian Wende, Christian Werner, Claas Wilke, Uwe Assmann
Technische Universität Dresden
D-01062, Dresden, Germany
Email: firstname.lastname@tu-dresden.de

***Abstract*—The history of all programming languages exposes the introduction of new language features. In the case of Java—a widespread general purpose language—multiple language extensions were applied over the last years and new ones are planned for the future. Often, such language extensions provide means to replace complex constructs with more compact ones. To benefit from new language extensions for large bodies of existing source code, a technique is required that performs the modernisation of existing source code automatically.**

In this paper we demonstrate, how Java programs can be automatically migrated to new versions of the Java language. Using JaMoPP, a tool that can create models from Java source code, we enable the application of model transformations to perform model-driven modernisation of Java programs. Our approach is evaluated by applying two concrete transformations to large open source projects. First, we migrate classical for loops to the new for-each style (introduced in Java 5). Second, we convert anonymous classes to closures (planned for Java 8). Furthermore, we discuss how tracing transformations allows to quantify the impact of planned extensions.

I. Introduction

Programming languages evolve over time: new features are added and occasionally old ones are removed. A prominent example of a language that undergoes such an evolution is Java. For example, generics were introduced in Java 5.

All changes—with a few exceptions—that were introduced to Java, preserved backward compatibility. Programs written in older versions do still compile and run with new versions. Still, old programs could be updated using new language features, assuming this improves code readability and therewith maintainability. For small programs, old code fragments can be replaced manually, but for large code bases automatic code modernisation transformations are required.

Source code transformations are known for quite a while and specialised tools exist to perform this task (cf. Sect. V). However, with the advent of Model-Driven Software Development (MDSD) [1], standardised transformation languages (e.g., Query View Transformation (QVT)[1]) became available. If one could use these languages for code transformation, the need for specialised languages would vanish.

To apply a model transformation language to source code, a model of the respective code is required. Also, a metamodel of the language that is subject to transformation is needed. In earlier work, we presented JaMoPP [2], [3]—the Java Model Printer and Parser—a tool that entails a complete metamodel for Java and tooling to convert Java source code to Eclipse Modeling Framework (EMF) [4] models and vice versa. Therefore, JaMoPP enables arbitrary EMF-based tools to work on Java programs.

In this paper, we show how JaMoPP and a standardised model transformation language can be combined to migrate Java code to a new version of the Java language. We present two concrete migration examples. First, existing for loops are transformed to the for-each style that was introduced in Java 5. Second, the conversion of anonymous inner classes to closures, which are planned for the Java 8 release, is performed. We apply the two transformations to a set of large open source Java projects. The results of this transformation can be used to quantify the impact of new language features.

The paper is structured as follows. After giving a brief overview on JaMoPP in Sect. II, we discuss the transformations for the two migration examples in Sect. III. The result of performing the transformations of larger bodies of source code can be found in Sect. IV. We compare our work with related approaches in Sect. V, and conclude in Sect. VI.

II. JaMoPP—Brief Overview

In MDSD, many generic modelling tools exist that can be used in combination with arbitrary languages. This is possible because the tools can be configured with metamodels. A metamodel describes the concepts of a language; also referred to as the abstract syntax of a language. To exchange metamodels between tools, the OMG has standardised the metamodelling languages Meta-Object Facility (MOF) and Essential Meta-Object Facility (EMOF)[2]. A widely used implementation of EMOF is Ecore as part of EMF. To use a generic modelling tool that supports Ecore with a certain language, a metamodel of that language needs to be provided together with tooling to parse (and print) sentences written in the language's concrete syntax (e.g., a Java program) into typed graphs that conform to the metamodel.

With JaMoPP, we provide such an Ecore metamodel and the tooling to parse and print source code for the Java language (currently supporting Java 5). This allows

[1] http://www.omg.org/spec/QVT/

[2] http://www.omg.org/spec/MOF/2.0

developers to apply generic modelling tools on Java source code and hence to use the same tools to work with models (e.g., UML models) and source code. An example of such a tool, which we show in the next section, is the QVT transformation language.

Important properties of JaMoPP are: (1) JaMoPP supports both parsing and printing Java code which allows modelling tools (e.g., model transformation engines) to both read and modify Java source code. This conversion preserves the layout of Java source code. (2) In addition to parsing, JaMoPP performs name and type analysis of the source code and establishes links (e.g., between the usage and definition of a Java class). These links can be exploited by modelling tools to ensure correctness of static semantics properties of the Java source files they generate or modify. (3) JaMoPP itself was developed using our modelling tool EMFText [5]. There, the concrete syntax of Java is defined in an EBNF-like syntax definition. Based on the metamodel and this definition, the parser and printer tooling is generated. This allows us to extend JaMoPP by extending the metamodel and the syntax definition without the need to modify code. With this, JaMoPP can co-evolve with future Java versions and can, in particular, be used to prototype and experiment with new features. An example of this is closure support, which is used in Sect. IV.

JaMoPP has been tested with a large body of source code from open-source Java projects through which stability and support for all Java 5 language features is assured (see [2] for details). Initially [2], we focused on using JaMoPP for forward engineering to generate and compose Java code. In [3] we presented how JaMoPP is used for reverse engineering. In the next section, we demonstrate how JaMoPP is used in combination with QVT for modernisation, which is a combination of reverse and forward engineering.

III. Model-driven Source Code Transformation

In this section we first exemplify Model-Driven Modernisation using our first migration example—the transformation of for loops to the for-each loops. Afterwards we discuss the benefits and challenges we experienced in applying model transformations for source code modernisation in both migration examples (for-each loops and closures). The complete transformation scripts can be found online[3].

To implement our modernisation transformations we used QVT-Operational provided by the Eclipse M2M project[4]. We consider the declarative language QVT-Operational a pragmatic choice for the unidirectional transformations typically required for source-code modernisation. In contrast, its declarative counterparts (QVT-Relations, QVT-Core) are more suitable for bi-directional transformation.

[3]http://jamopp.org/index.php/JaMoPP_Applications_Modernisation/
[4]http://www.eclipse.org/m2m/

```
1 mapping transformForLoopToForeachLoop
2   (forLoop : JAVA::statements::ForLoop)
3   : JAVA::statements::ForEachLoop
4   when {
5     forLoop.checkLoopInit() and
6     forLoop.checkLoopCondition() and
7     forLoop.checkLoopCountingExpression() and
8     forLoop.checkLoopStatements()
9   }{
10  var listType : JAVA::types::TypeReference
11    := getIteratedCollectionType(counterIdentifier);
12  var loopParameter
13    := object JAVA::parameters::OrdinaryParameter {
14      name := "element";
15      typeReference := listType };
16  result.next := loopParameter;
17  result.statement :=
18    map replaceCollectionAccessorStatements(
19      forLoop, loopParameter);
20 }
```

Listing 1. QVT Transformation to replace for loops with for-each loops.

A. Example of Model-Driven Modernisation of For Loops

Listing 1 shows a mapping taken from the transformation of for loops to for-each loops. An example of the expected replacement and a description of the loop elements used in the transformation is given in Fig. 1. The mapping consists of a when-clause (lines 4-9) and a mapping body (lines 10-20). The when-clause defines a number of preconditions that need to be satisfied by a given for loop to be replaced. For instance, that the `init-statement` initialises the loop `counter variable` with `0`, that the loop `condition` ensures iteration among all collection elements, or that the `counting expression` always increases the loop `counter variable` by 1. Furthermore, the `loop statements` are not allowed to refer to the `counter variable` aside from accessing the collection for the next element. If all preconditions are satisfied we calculate the generic type of the collection that is iterated (lines 10-11) and create an `iteration parameter` for the for-each loop that is initialised with this type (line 12-15). Finally, the new for-each loop is initialised with this parameter and its body is filled with the statements of the original for loop, where all statements for `collection access` are replaced with `references to the iteration parameter` of the for-each loop.

B. Applicability of QVT for Model-Driven Modernisation

For the specification of both transformation scripts we used a set of tools provided by the M2M project for QVT-Operational. The included editor provided advanced editing features like syntax highlighting, code navigation, and code completion. Especially code completion helped a lot in writing expressions that traverse and analyse Java models. A second tool that helped a lot in developing transformations was the QVT debugger. It allows the step-wise evaluation of transformation execution and was indispensable to understand and fix problems in our complex transformation scripts. Third, the QVT interpreter generates

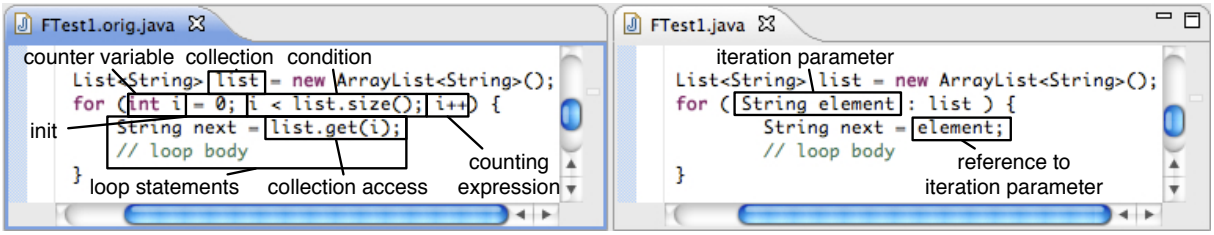


Figure 1. Example of for loop replacement explaining the elements of for- and for-each loops.

tracing information for each execution of the transformation. The trace records all mappings applied and enabled the quantitative analysis of our examples. We think that the reusability and maturity of these generic tools provide some good arguments for applying a standardised transformation language.

A benefit of model-driven modernisation was the graph-structure of models, which, compared to tree-structures often provided by code parsing tools, allow for more convenient navigation and analysis of references between declarations and uses of code elements (e.g., variables). For example, this eased the specification of the preconditions for the for loop migration that searches the method body for statements that use the counter variable declared in the loop header.

In both examples it is not trivial to come up with an exhaustive set of patterns that identify source code that can be modernised. We consider this a challenge for source code modernisation in general. However, we also learned that some idioms (like the for loop presented in Fig. 1) are quite common and occurred in all Java projects we investigated, as can be seen in the next section.

Some drawback of using model transformations for code modernisation was the focus on abstract syntax, i.e., the language metamodel. It required a good knowledge of the JaMoPP metamodel and some training to represent patterns of source code in abstract syntax. On the other hand, the strict structure of an explicit metamodel was beneficial to ensure the well-formedness of the produced source code.

IV. Evaluation

To evaluate the performance of our source code modernisation approach, we applied the transformations from Sect. III to a set of Java frameworks available to the public. Our goal was to answer the following questions: First, we wanted to know whether a general purpose modelling environment like EMF is scalable enough to handle such a large set of models. Second, we were curious how many resources are required to perform a transformation of this scale with QVT—a generic model transformation language. To answer these questions, we transformed 16.402 Java files from 10 open source projects.

A. Performance

To evaluate the transformation performance, we measured the time needed to perform the transformations on individual compilation units. This includes all types referenced by this unit, but excludes other, unrelated parts of the source code. We used a machine with a Dual Core AMD Opteron running at 2.4 GHz with 4 GB RAM. We used only one core of the machine to avoid problems with Eclipse plug-ins that are not thread-safe.

Framework	Files	For loops		Closures	
		min	repl./occur.	min	repl./occur.
AndroMDA 3.3	698	3	4/959	8	8/201
Apache Ant 1.8.1	829	5	24/1028	21	12/99
Comns. Math 1.2	395	1	25/845	5	0/25
Tomcat 6.0.18	1127	4	65/1437	15	52/125
GWT 1.5.3	1850	5	29/1044	23	26/670
JBoss 5.0.0.GA	6414	16	472/2744	70	197/591
Mantissa 7.2	242	1	7/652	3	18/29
Spring 3.0.0.M1	3096	8	82/680	43	31/1403
Struts 2.1.6	1035	3	7/130	13	8/158
XercesJ 2.9.1	716	3	21/1111	12	62/94
	16402	49	736/10630	213	414/3395

Figure 2. Transformation time and ratio of found and replaced elements.

The results of our measurements are shown in Fig. 2. From these numbers one can derive that the pure average transformation time per source file is 0.2 seconds (for-each loops) and 0.8 seconds (closures). These values can be obtained by dividing the total time (given in minutes in columns 3 and 5) by the number of total source files. In addition to the transformation time one must also take into account the time needed to load the input models. This can take up to a few minutes for very complex source files, but is usually done within few seconds. For a migration task, which is performed once for every new release of a programming language, this renders the approach still feasible.

B. Quantification of Language Extensions

To quantify the impact of a planned language extension, one can count the number of replaceable language constructs. This is usually only a subset of all cases where a new language construct is applicable. Some potential applications of a language construct may simply not be detected, because developers used structures not covered in the transformation script.

Nonetheless, the number of places in existing source code where a new language construct can be applied, does at least give some indication about its impact. The ratio for loops that can be replaced by for-each loops to all for loops found, and the ratio of anonymous inner classes that are replaceable by closures to all inner classes found is shown in Fig. 2.

On average, 6.9% of all for loops were transformed to the for-each style. The percentage of anonymous classes that were replaced by closures was 12.2%. Certainly, these numbers can be increased if more patterns are covered by the transformation scripts. However, given the very restrictive scripts which we used, the numbers are quite high. Thus, one can reason that actual benefit can be gained by using automatic transformations here and that both language extensions are useful additions to the Java language.

V. Related Work

There exists a large amount of work and tools for source code transformation.[5] One prominent approach is Stratego/XT [6], a tool set for strategic rewriting of source code. While Stratego/XT and other approaches are proved to be useful and applicable in academic and industrial projects, they do not provide a standardised transformation language. With JaMoPP one can transform source code to a standardised intermediate representation (i.e., EMF-based models) where we apply a standardised transformation language (i.e., QVT). This can be generalised to other programming languages and other transformation languages.

MoDisco [7] aims at discovering models from legacy applications. It handles Java code as well as other artefacts (e.g., configuration files). Based on the Eclipse JDT parser, MoDisco creates models from source code files. The metamodel for these models is defined in Ecore similar to JaMoPP. Thus, transformations can be specified using existing languages (e.g., QVT). However, MoDisco does not preserve the layout of the source code when printing back transformed models back to their original representation (i.e., Java source code). Approaches for metamodel evolution (e.g., [8]) are inherently limited to the model level and do not allow to print models after performing evolution steps.

Architecture-driven Modernisation (ADM)[6] of the OMG goes beyond what is presented in this paper by applying modernisation efforts on all artefacts involved in the software development process (e.g., source code, database definitions, configuration files, ...). However, strategies that are built on top of existing OMG standards (e.g., similar to the combination of EMOF and QVT in our approach) can fit nicely in the overall goal of ADM.

VI. Conclusion & Future Work

In this paper, we implemented and evaluated two examples of Java modernisation to show that JaMoPP in combination with the standardised model transformation language QVT can be used for Model-Driven Modernisation of Java programs. In applying the transformations on a set of open-source projects we experienced that the transformations can be performed in acceptable time and transformed a reasonable part of the code (on average, 6.9% of all for loops and 12.2% of all anonymous classes were considered as candidates for transformation). So far we only used our own judgement based on our experience with Java to determine the semantic correctness of the transformation rules. In future we plan to automatically validate correctness by running the test suites of the open-source projects on the modernised code. Further, we did not yet compare the effort of writing QVT transformations for Java with alternatives such as using Java-specific source code transformation tools or other model transformation languages. Doing this comparison by implementing the two modernisation transformations with different languages and tools is subject to future work.

[5] http://www.program-transformation.org/ provides an overview.

[6] http://adm.omg.org/

Acknowledgment

This research is co-funded by the European Commission within the projects MODELPLEX #034081 and MOST #216691, by the German Ministry of Education and Research (BMBF) within the projects feasiPLe and SuReal; by the German Research Foundation (DFG) within the project HyperAdapt and by the European Social Fund and Federal State of Saxony within the project ZESSY #080951806.

References

[1] M. Völter, T. Stahl, J. Bettin, A. Haase, and S. Helsen, *Model-Driven Software Development*. John Wiley & Sons, 2006.

[2] F. Heidenreich, J. Johannes, M. Seifert, and C. Wende, "Closing the Gap between Modelling and Java," in *Proc. of 2nd Int'l Conf. on Software Language Engineering (SLE'09)*, ser. LNCS, vol. 5969. Springer, Mar. 2010, pp. 374–383.

[3] ——, "Construct to Reconstruct—Reverse Engineering Java Code with JaMoPP," in *Proc. of Int'l Workshop on Reverse Engineering Models from Software Artifacts (R.E.M. 2009)*, Oct. 2009.

[4] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *Eclipse Modeling Framework (2nd Edition)*. Pearson Education, 2009.

[5] F. Heidenreich, J. Johannes, S. Karol, M. Seifert, and C. Wende, "Derivation and Refinement of Textual Syntax for Models," in *Proc. of ECMDA-FA'09*, ser. LNCS, vol. 5562. Springer, Jun. 2009, pp. 114–129.

[6] E. Visser, "Program transformation with Stratego/XT: Rules, strategies, tools, and systems in StrategoXT-0.9," in *Domain-Specific Program Generation*, ser. LNCS, C. Lengauer, D. Batory, C. Consel, and M. Odersky, Eds. Spinger, Jun. 2004, vol. 3016, pp. 216–238.

[7] H. Bruneliere, J. Cabot, F. Jouault, and F. Madiot, "MoDisco: A Generic and Extensible Framework for Model Driven Reverse Engineering," in *Proc. of ASE'10*. ACM, 2010, pp. 173–174.

[8] M. Herrmannsdoerfer, S. Benz, and E. Juergens, "COPE - Automating Coupled Evolution of Metamodels and Models," in *Proc. of ECOOP'09*, ser. LNCS, S. Drossopoulou, Ed., vol. 5653. Springer, 2009, pp. 52–76.

DynaMod Project: Dynamic Analysis for Model-Driven Software Modernization

André van Hoorn*, Sören Frey*, Wolfgang Goerigk†, Wilhelm Hasselbring*, Holger Knoche†, Sönke Köster§, Harald Krause‡, Marcus Porembski§, Thomas Stahl†, Marcus Steinkamp§, and Norman Wittmüss‡

*Software Engineering Group, University of Kiel, Christian-Albrechts-Platz 4, 24098 Kiel
†b+m Informatik AG, Rotenhofer Weg 20, 24109 Melsdorf
‡Dataport, Altenholzer Straße 10-14, 24161 Altenholz
§HSH Nordbank AG, Schloßgarten 14, 24103 Kiel

***Abstract*—Our project DynaMod addresses model-driven modernization of software systems. Key characteristics of the envisioned approach are: (1) combining static and dynamic analysis for extracting models of a legacy system's architecture and usage profile; (2) augmenting these models with information that is relevant to the subsequent architecture-based modernization steps; and (3) automatically generating implementation artifacts and test cases based on the information captured in the models. This paper provides an overview of the DynaMod project.**

I. Introduction

Outdated programming technologies or platforms are typical reasons for long-lived software systems to age. Continuous and sustainable modernization is required for retaining their maintainability.

In our project DynaMod, we will investigate techniques for model-driven modernization (MDM) of software systems. Innovative aspects are the combination of static and dynamic analysis for reverse engineering architectural and usage models, as well as its semantic augmentation by information supporting subsequent generative forward engineering steps, including tests. DynaMod started with the beginning of 2011 and has a two-year funding from the German Federal Ministry of Education and Research (BMBF).

The remainder of this paper provides a brief summary of the envisioned approach (Section II), the consortial partners (Section III), and the working plan (Section IV).

II. Envisioned DynaMod Approach

The envisioned approach can be structured into three phases:

1) Extracting architecture-level models from the outdated system and its usage.
2) Defining architecture-based transformations to models of the desired system.
3) Generating implementation artifacts and tests for the modernized system.

Model extraction will be achieved by combining static and dynamic analysis, as well as additional augmentation. Static analysis of the source code yields a mainly structural view, e.g., including architectural entities and relations. Dynamic analysis, comprising continuous monitoring, adds quantitative runtime behavior information, such as workload characteristics and execution frequencies of code fragments. The models are further refined by manually augmenting them with information that can only be provided by system experts, e.g., mapping code fragments to architectural layers. The conceptual modernization is performed on the architectural level by defining transformations among the extracted models of the outdated system and models of the target architecture. Established generative techniques from model-driven software development (MDSD) [1] are utilized to develop the modernized system. Test cases are generated by including the usage profile information from the dynamic analysis. Further evolution of the modernized systems is based on the MDSD paradigm, keeping the models synchronized with the system implementation.

The DynaMod project is funded by the German Federal Ministry of Education and Research under the grant numbers 01IS10051A and 01IS10051B.

III. Project Consortium

The DynaMod project consortium consists of the b+m Informatik AG (development partner and consortium leader), the University of Kiel (scientific partner), as well as two associated companies, Dataport and HSH Nordbank AG.

- *b+m Informatik AG:* The b+m group offers IT solutions, including consultancy, software engineering, and maintenance services. Being the initiator of the openArchitectureWare (oAW) framework, b+m is known for its pioneering role in developing and applying MDSD techniques and tools.
- *University of Kiel, Software Engineering Group:* The Software Engineering Group conducts research in the area of software engineering for parallel and distributed systems. One focus is the investigation of model-driven techniques and methods for engineering, operating, as well as evolving software systems, having an emphasis on the consideration of software quality.
- *Dataport:* Dataport provides information and communication technology services for the public administration in the German federal states of Schleswig-Holstein, Hamburg, Bremen, and for the tax administration of the federal states of Mecklenburg-Vorpommern and Niedersachsen.
- *HSH Nordbank AG:* HSH Nordbank is a leading bank for corporate and private clients in northern Germany. As one of the major providers of real estate finance in Germany it focuses on serving commercial clients. In the regionally rooted key industries of shipping, aviation and energy & infrastructure the bank also operates internationally as a top provider of finance solutions.

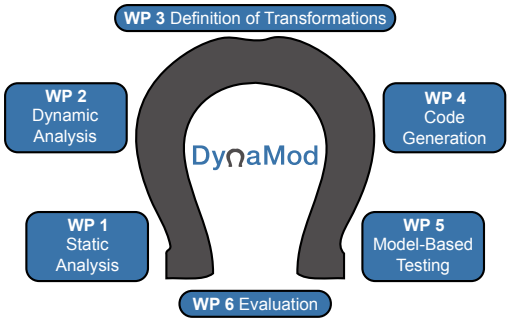


Fig. 1. DynaMod work packages—aligned with the horseshoe model

IV. Working Plan

Based on the envisioned approach outlined in Section II, some more details about our working plan, including involved technologies and desired results, are described in this section. The working plan is structured into six technical work packages (WPs) which are aligned with the horseshoe model for re-engineering, as depicted in Figure 1.

WP1—Static Analysis: This work package investigates methods for the extraction of architectural models utilizing static analysis techniques. Therefore, we will define appropriate meta-models on the basis of the OMG's Architecture-Driven Modernization (ADM) standards, e.g., KDM and SMM. A specific challenge can be seen in providing an adequate representation, abstraction level, and semantic meta-data describing the outdated system to enable smooth integration with information obtained from dynamic analysis (see WP 2). Parsers are required to extract the models from the source code. We will limit ourselves to the programming platforms emerging from our case study scenarios (WP 6).

WP 2—Dynamic Analysis: This work package is concerned with the dynamic analysis of a legacy system's internal behavior and external usage profile while being deployed in its production environment. The instrumentation of systems relying on legacy technology constitutes a technical challenge to be tackled. We plan to extend our monitoring analysis framework Kieker [2], currently restricted to Java-based systems, to support additional programming platforms introduced by the case studies (WP 6). An interesting research question to be addressed here is to identify which information is relevant to the MDM process.

WP 3—Definition of Transformations: The enriched architecture-level models describing the existing system are being translated towards a target architecture employing model-based transformation techniques. Therefore, the corresponding transformation rules are created in this work package. Regarding a first transformation type, KDM may serve as a meta-model for both the source and the target models. A second type transforms these architecture models in code-centric instances. Here, classifying the transformations and exploring transformation patterns can improve the efficiency of the modernization process.

WP 4—Code Generation: Based on MDSD model-to-code transformation techniques, we will use templates to generate implementation artifacts, e.g., SOA wrappers and connectors, from the models of the target architecture (WP 3). The tooling infrastructure will be based on the Eclipse Modeling Project (EMP) which includes the former oAW framework. In this work package, we will focus on the code generation for the case study scenarios (WP 6).

WP 5—Model-Based Testing: Usage models extracted from the legacy system under production workload (WP 2) will be used to generate representative test cases which can be executed automatically by appropriate testing tools. We will focus on tests based on workload generation, allowing to compare quality properties, such as performance and reliability, among the modernized and the outdated system. It is intended to develop transformations from models to test plans to be executed by the load test tool Apache JMeter as well as the extension Markov4JMeter, supporting probabilistic models.

WP 6—Evaluation: In addition to lab studies, the developed methodology and tooling infrastructure will be evaluated based on the three below-described case study systems from the associated partners. Most likely, these systems will not be completely modernized during the DynaMod project. We consider them to be benchmark examples and representatives of modernization projects pending in practice.

- *AIDA-SH (Dataport)* is an information management and retrieval system for inventory data of historical archives. The system conforms to a client/server architectural style, largely based on Microsoft technology: database management systems (DBMSs) based on MS SQL Server (7.0, 2000, and 2003) as well as MS SQL Desktop Engine (MSDE), and a user interface implemented with Visual Basic 6 (VB 6). The major impulse for modernization is the outdated technology.
- *Nordic Analytics (HSH Nordbank)* is a function library for the assessment and risk control of finance products. The library is used in desktop installations (from an Excel front-end), and, deployed to a grid infrastructure, by online trading and batch processing systems. Nordic Analytics is implemented using C#. A modernization will focus on architectural restructuring.
- *Permis-B (Dataport)* is a mainframe system for managing health care allowance. The technical platform consists of z/OS (mainframe operating system), ADABAS-C (DBMS), COMPLETE (transaction processing monitor), as well as the programming environments NATURAL and COBOL. User interfaces are provided by an EskerTun/HobLink terminal emulation and a Web interface (supporting only MS IE 7) based on VB 6 and MS SQL Server 2003. Desire for modernization is pushed by the outdated technology and an eroded architecture, making it difficult to fulfill additional or changed functional requirements in the future.

References

[1] T. Stahl and M. Völter, *Model-Driven Software Development – Technology, Engineering, Management*. Wiley & Sons, 2006.

[2] A. van Hoorn, M. Rohr, W. Hasselbring, J. Waller, J. Ehlers, S. Frey, and D. Kieselhorst, "Continuous Monitoring of Software Services: Design and Application of the Kieker Framework," Dept. Comp. Sc., Univ. Kiel, Germany, Tech. Rep. TR-0921, 2009.

REMICS Project: Reuse and Migration of Legacy Applications to Interoperable Cloud Services

Parastoo Mohagheghi, Arne J. Berre
SINTEF, Norway
{Parastoo.Mohagheghi, Arne.J.Berre}@sintef.no

Franck Barbier
University of Pau - Netfective Technology, France

Alexis Henry
BLU AGE Software - Netfective Technology, France

Andrey Sadovykh
SOFTEAM, France

***Abstract*—The main objective of the REMICS project is to specify, develop and evaluate a tool-supported model-driven methodology for migrating legacy applications to interoperable service cloud platforms. The migration process consists of understanding the legacy system in terms of its architecture and functions, designing a new SOA application that provides the same or better functionality, and verifying and implementing the new application in the cloud. The demonstrations will show the support for two tasks in this migration: recovery process with the BLU AGE tool and the use of SoaML and forward engineering with Modelio tool.**

I. REMICS Approach and Demonstrations

The REMICS[1] project will provide tools for model-driven migration of legacy systems to loosely coupled systems following a bottom up approach; from recovery of legacy system architecture (using OMG's ADM-Architecture Driven Modernization) to deployment in a cloud infrastructure which allows further evolution of the system in a forward engineering process. The migration process consists of understanding the legacy system in terms of its architecture, business processes and functions, designing a new Service-Oriented Architecture (SOA) application, and verifying and implementing the new application in the cloud. These methods will be complimented with generic "Design by Service Composition" methods providing developers with tools simplifying development by reusing the services and components available in the cloud.

In order to instrument the migration process, the REMICS project will integrate a large set of metamodels and will propose several dedicated extensions. For the architecture recovery the REMICS will extend the KDM metamodel. On Platform Independent Model (PIM) level, the components and services are defined using SoaML (SOA Modeling Language[2]) which is developed in the SHAPE project[3]. The REMICS project will extend this language to address the specific architectural patterns and model driven methods for architecture migration, and to cover specificities of service clouds development paradigm. In particular, the PIM4Cloud Computing, model-driven Service Interoperability and Models@Runtime extensions are intended to support the REMICS methodology for service cloud architecture modeling.

Furthermore, REMICS will investigate existing test notations such as the UML2 test profile (UTP) for their application to the SOA and Cloud Computing domain and refine and extend them.

The project will focus on open source metamodels and models with an emphasis on Open Models for standards and will be actively involved in the standardization process of the related standards for cloud computing, business models, SOA, service interoperability, knowledge discovery, validation and managing services.

REMICS targets the following main impact objectives:

- REMICS will preserve and capitalize on the business value engraved in legacy systems to gain flexibility brought by Service Clouds, lower the cost of service provision and shorten the time-to-market.
- REMICS research will provide innovations in advanced model driven methodologies, methods and tools in Software as a Service engineering.
- REMICS will provide standards-based foundation service engineering and will provide a suite of open ready-to-use metamodels that lowers barriers for service providers.

REMICS started in September 2010 and will run for three years while it builds on the results of several ongoing or finished EU projects such as SHAPE and MODELPLEX[4] (both finished recently) with focus on model-driven development of applications, MOMOCS with focus on model-driven modernization, and SOA4ALL and RESERVOIR with focus on service-oriented development. The relevant results of previous projects will therefore be discussed and extensions planned in REMICS will be presented. The presentation will also discuss collaboration areas which should be of interest to other projects and conference participants.

The demonstrations will show the support for two tasks in this migration: recovery process using BLU AGE[5] tool and the use of SoaML and forward engineering with Modelio[6] tool. Both tool providers are participating in the REMICS project.

[1] http://remics.eu/; funded by the European Commission (contract number 257793) within the 7th Framework Program

[2] http://www.omg.org/spec/SoaML/

[3] http://www.shape-project.eu/

[4] https://www.modelplex-ist.org/

[5] http://www.bluage.com/; a solution for both reverse and forward engineering fully based on MDA and ADM principles

[6] http://www.modeliosoft.com/

SOAMIG Project: Model-Driven Software Migration towards Service-Oriented Architectures

A. Winter, C. Zillmann
OFFIS Oldenburg
Institute for Information Technology
{zillmann, winter}@offis.de

A. Fuhr, T. Horn, V. Riediger
Institute for Software Technology (IST)
University of Koblenz-Landau
{afuhr, horn, riediger}@uni-koblenz.de

A. Herget, W. Teppe, M. Theurer
Amadeus Germany
{aherget, wteppe, mtheurer}@de.amadeus.com

U. Erdmenger, U. Kaiser, D. Uhlig, Y. Zimmermann
pro et con GmbH
{uwe.erdmenger, uwe.kaiser, denis.uhlig, yvonne.zimmermann}@proetcon.de

***Abstract*—The SOAMIG project aims at developing a general migration process for model-driven migrations towards Service-Oriented Architectures. This paper highlights the model-driven tools developed during the SOAMIG project for two case studies: A language migration from a COBOL transactional server to Java web services, and a second study on an architecture migration from a monolithic Java fat client to a SOA-based JavaEE web application.**

I. Motivation

Today, companies are facing a growing competition in their markets. Competitors are forced to achieve higher flexibility and faster time-to-market in order to survive. Often, so-called legacy software developed in the companies can not keep up with this highly dynamic environment and therefore slows down innovation.

For this reason, companies are looking for flexible software concepts supporting fast adaptability to business changes. A promising approach to achieve the required flexibility are Service-Oriented Architectures (SOAs). SOAs encapsulate functionality in coarse-grained, loosely-coupled and reusable units, called services.

Adopting SOAs, companies do not want to throw away their existing systems because much money and knowledge has been put into them. Instead of reimplementing the service functionality from scratch, companies are striving to reuse their legacy software as much as possible. Transferring existing code into a new technology without changing functionality is called software migration.

The SOAMIG project, partially funded by the German Ministry of Education and Research (BMBF)[1], brings together both: transition into SOA by migrating the legacy code. The overall goals are i) to define a reference process [1], ii) to achieve a high degree of automatic code migration, and iii) to support the migration process by analysis and transformation tools.

In this project, two universities and two companies have been involved: the *Universities of Oldenburg* (OFFIS) and *Koblenz-Landau* (IST) supplying reengineering knowledge and model-driven tools, *pro et con*, supplying long-time expertise in industrial migration projects, language analysis, and migration tools development, and *Amadeus Germany*, providing one of the industrial legacy systems and know-how in migration of large-scaled systems.

[1]Grant no. 01IS09017A-D. See http://www.soamig.de for further information.

During the SOAMIG project, two industrial case studies were selected: the *LCOBOL* case study deals with a language migration from a transaction driven COBOL system to Java Web Services, while the *ASPL* case study is about an architecture migration from a Java fat client into a Java SOA. This short-paper presents the tools developed during the SOAMIG project.

II. LCOBOL: Language Migration

The LCOBOL case study is conducted by pro et con, one of the industrial partners. The main challenge in this case study is to yield a very high degree of automation for a language migration from COBOL to Java. Also, the resulting Java code has to be understandable and maintainable. Figure 1 shows the set-up of the tool chain.

Every COBOL source file is parsed into a fine-grained abstract syntax graph by the COBOL front-end `CobolFE`. `CobolFE` can handle various COBOL dialects. The main translation to Java is done by the model-to-model transformation `Cobol2Java`. This tool takes the COBOL model as input. The actual transformation is defined by many sophisticated rules defining a semantics-preserving transformation into a Java model. Project specific rules, e.g., on how to transform specific transaction monitor calls, amend the language translation. The transformations are implemented in C++. Finally, Java source code is generated by `JGen` and `JFormat`. `JGen` takes a model of a Java translation unit as input and creates syntactically correct, but only roughly formatted output. `JFormat` is a stand-alone, scriptable Java source code formatter based on the Eclipse JDT formatter and is individually configurable to various formatting conventions.

III. RailClient: Architecture Migration

In the RailClient case study, an architecture migration from a monolithic Java system into a SOA-based web application is investigated by all four project partners. Figure 2 outlines the tools developed in this part of the SOAMIG project. Amadeus Germany provided the business case and the subject system, an order management and booking system for train tickets. OFFIS contributed to the definition and realization of the target architecture.

The *SOAMIG repository* forms a common core of the tool-chain. In this repository, artifacts used during migration are stored as models. The main part of the repository

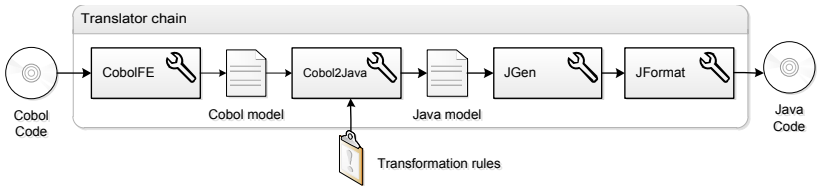


Figure 1. Tool set-up for the LCOBOL case study

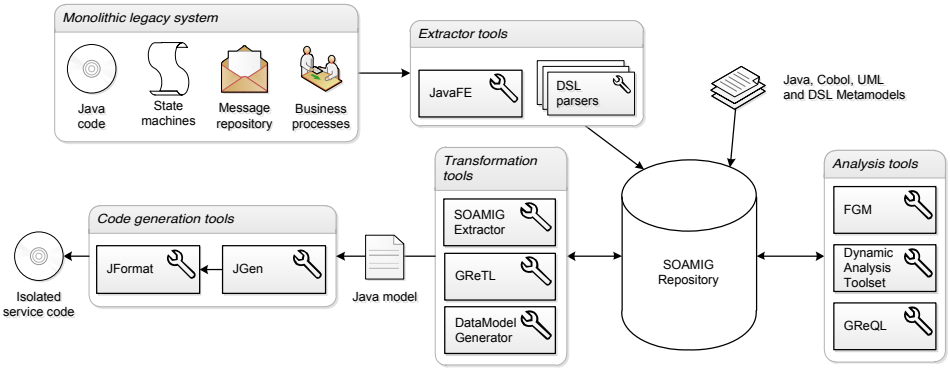


Figure 2. Tool set-up for the RailClient case study

is based on the TGraph technology developed by IST. TGraphs are a versatile data structure formally defined by grUML (graph UML). The TGraph technology is generic and can represent arbitrary artifacts. In SOAMIG, the tools are integrated by an XML-based exchange format for metamodels and models.

To ease initial program understanding and redocumentation, the explorative tool `FGM` (Flow Graph Manipulator) by pro et con was used. This partner also provided `JavaFE`, an extractor from Java source to fine-grained abstract syntax graphs stored in the repository. Not only source code, but also various other parts of the legacy system, such as automatons controlling GUI behavior, message descriptions, and redocumented business processes are combined into one comprehensive model. Links between these parts are established by static and dynamic analysis. Static analysis is covered by `JavaFE` and `GReQL` (Graph Repository Query Language). Dynamic analysis of certain test cases covering the selected business processes is used to detect relevant portions of the source code, and to mark service candidates [2].

Among the transformation tools in this case study is `GReTL` (Graph Repository Transformation Language), a general-purpose transformation language that allows to define and execute arbitrary graph transformations. A system specific `DataModelGenerator` combines message descriptions and dynamic traces to compile service specific data structures for the target architecture.

The `SoamigExtractor` tool provides a graphical interactive user interface to enable model transformations. Examples are incorporating dynamic traces, pruning generalization hierarchies, selection and completion (slicing) of multi-class Java models based on execution traces, establishment of traceability links between source and target models, and export of translation unit models to the above mentioned `JGen` and `JFormat` tools. The generated Java code requires manual rework. It contains human readable as well as machine processable annotations to link to the relevant legacy sources.

IV. Conclusion

Summarizing, a set of powerful model-driven tools and technologies has been developed to support various tasks during the migration process. Most of the tools are independent of the concrete legacy system and are reusable as-is, others have to be configured or have even been built from scratch. The SOAMIG repository technology is largely generic, enabling integration of additional metamodels and traceability to the already existing parts.

Every migration project requires adaption and specialization of process, repository, and tools to the specific needs of the legacy and target systems, the organizational requirements, and other factors. While the complete migration of the case study systems is out of scope of the project, the model-driven tools have proven to be applicable in real-world scenarios. Transfer to other business cases and different migration tasks is an opportunity to further evolution of the SOAMIG process and technology.

References

[1] U. Erdmenger, A. Fuhr, A. Herget, T. Horn, U. Kaiser, V. Riediger, W. Teppe, M. Theurer, D. Uhlig, A. Winter, C. Zillmann, and Y. Zimmermann, "The SOAMIG Process Model in Industrial Applications," in *Proceedings of the 15th European Conference on Software Maintenance and Reengineering*, T. Mens, Y. Kanellopoulos, and A. Winter, Eds. Los Alamitos: IEEE Computer, 2011, pp. 339–342.

[2] A. Fuhr, T. Horn, and V. Riediger, "Dynamic Analysis for Model Integration (Extended Abstract)," *Softwaretechnik-Trends*, vol. 30, no. 2, pp. 70–71, 2010.

Comprehensive Model Integration for Dependency Identification with EMFTrace*

Stephan Bode, Steffen Lehnert, Matthias Riebisch
Department of Software Systems / Process Informatics
Ilmenau University of Technology
Ilmenau, Germany
{*stephan.bode, steffen.lehnert, matthias.riebisch*}*@tu-ilmenau.de*

***Abstract*—As model-based software development becomes increasingly important, the number of models to express various aspects of software at different levels of abstraction raises. Meanwhile evolutionary development and continuous changes demand for explicit dependencies between involved models to facilitate change impact analysis, software comprehension, or coverage and consistency checks. However, there are no comprehensive approaches supporting models, dependencies, changes, and related information throughout the entire software development process. The approach presented in this paper provides a unified and model-spanning concept with a repository for model integration, model versioning, and dependency identification among models utilizing traceability techniques, enhanced with analytic capabilities. The identification is based on a rule set to provide high values for precision and recall. The approach is implemented in a tool called EMFTrace, which is based on Eclipse technology and supports different CASE tools for modeling.**

***Keywords*-model dependencies; model integration; model repository; meta model; traceability**

I. Introduction

The growing complexity of software is often accompanied by difficulties in implementing changes and by architectural erosion. One possibility to tackle these problems is the use of different models and modeling languages throughout the entire development process to express requirements, design decisions, and dependencies between models, which is commonly referred to as model-based development. The wide use of several modeling languages and CASE tools has led to a variety of models with different scope and levels of abstraction. Integrating the various models and detecting and analyzing dependencies between them has become important.

It is therefore necessary to provide support for software evolution and continuous changes through models at different levels of abstraction and by explicitly expressing dependencies between them. Traceability links are well-suited to model such dependencies, since they enable further analyses, as change impact analysis, software comprehension, and consistency checks. Moreover, solid tool support is required to cope with the number of modeled entities and dependencies to supply practical applicability and usability.

Existing approaches provide solutions for the integration of models into centralized repositories, e.g., [6]. Besides, several different concepts have been proposed for dependency analyses on models through traceability rules, e.g., in [1]–[4], whereas [5] relies on information retrieval. However, these approaches only support a limited number of modeling languages without presenting a solution for spanning the whole software development process. They lack a unified concept for managing models, dependencies between models, types of dependency relations and auxiliary information in a centralized manner. This unified concept should be enforced to span the entire development process, ranging from early requirement definitions up to regression testing with the help of appropriate models. Heterogenous development environments and the wide use of several different CASE tools amplify the benefits gained from a unified and process-spanning approach.

The concept presented in this paper is aimed at creating an extensible platform for various analyses techniques required for model-driven engineering, reengineering, and impact analysis to support software evolution and continuous changes. It is based on a unified approach for comprehensive model integration and dependency analysis. In our approach inter and intra model dependencies are identified and recorded through traceability links, which are established by explicitly defined rules. These traceability rules are combined with information retrieval algorithms to achieve high precision and recall as well as flexibility. Special emphasis has also been put on the explicit modeling of dependency types, since they provide important semantic information for further analyses. Moreover, our concept provides a unified treatment of all related models and data through automated versioning and management of models, rules, model dependencies, and dependency types within a centralized repository. We utilize the extensible model repository EMFStore [6], which supports the integration of arbitrary models of standardized modeling languages such as UML, URN [7], BPMN and OWL. Several analysis components were integrated into EMFTrace to validate dependencies and to facilitate further analysis capabilities.

The remainder of this paper is organized as follows. We discuss current development and research regarding model repositories and model dependencies in Section II, whereas Section III introduces our concept and describes our approaches for model integration and dependency analysis. Section IV outlines our main conclusions and future work.

*The research presented in this paper was partly funded by the federal state Thuringia and the European Regional Development Fund ERDF through the Thüringer Aufbaubank under grant 2007 FE 9041.

II. State of the Art

A. Model Repositories

Automated model management and versioning can be achieved by model repositories that therefore assist model-based development of software. We considered several different repository projects as a base for EMFTrace, including the Eclipse Model Repository[1], CDO Model Repository[2], EMFStore[3] and AndroMDA[4]. Maturity, supported features, usability, and documentation were the main decisive factors we based our evaluation on. After a careful consideration of all projects, we decided to use the EMF[5]-based EMFStore as our underlying repository.

B. Detection of Model Dependencies

Since we focus on model-based development and dependencies between models, we do not consider approaches for dependency detection on the level of source code. Data gained through dependency analyses can either be stored and maintained in a dependency matrix, as a graph of traceability links, or as a set of cross-references [8]. Traceability links allow to attach auxiliary information to them, such as design decisions and alternatives, or link types. This information is of an equivalent value as the actual dependency itself, since it increases the semantical meaning of a dependency relationship. Therefore, it is most suitable to represent model dependencies with traceability links for further analyses.

Most traceability approaches focus either on predefined rules or on information retrieval. Information retrieval approaches as proposed in [5] or [9] operate on identifiers and are therefore easy to adopt to new models and data. They provide good recall, but their lack of precision and the complete absence of additional dependency information limit their usability for dependency detection among models and further analyses.

In contrast, rule-based approaches require more work to adopt to new models, but they result in more reliable links. Rule-based approaches as proposed in [1]–[4] are able to provide auxiliary information on dependencies, such as the type of the dependency specified by the rule. Well-defined dependency types are inevitable to identify semantically rich dependencies and to classify them properly.

III. EMFTrace

A. Concept and Features

EMFTrace implements our approach of comprehensive model integration and dependency identification as an extensible platform, which is our basis for further analyses and research. Our main focus is to provide a set of analyses and validation features, which is independent of a certain CASE tool and modeling language, to support engineers, software architects and other stakeholders throughout the entire software lifecycle. Based on models and dependencies between them, we want to establish solid tool support for the following activities:

- Architecture comprehension,
- Dependency analysis,
- Change impact analysis,
- Goal-oriented decision support for:
  - Forward engineering and
  - Reengineering; as well as
- Validation and consistency checks.

We utilize the EMFStore model repository for storing and versioning EMF-based models. Figure 1 provides an overview of the entire approach and illustrates the interaction of EMFTrace with external CASE tools.

To enable the integration and dependency analysis of different models we extended EMFStore by several metamodels as shown in the bottom of Figure 1. We integrated different modeling languages into the repository to span the entire software development process. Our approach supports requirements engineering through URN goal models, URN use case maps, and UML use cases. Furthermore, design models such as class diagrams are provided by UML, which has been fully integrated as well. Additional dependencies between UML and URN models are provided by the integration of OWL ontologies, which supply semantic networks of related terms. To ensure practical applicability, we support several different CASE tools which can be used to create and edit the corresponding models. More details on the integration of the models are given in Section III-B2.

As mentioned in Section II our approach utilizes rule-based traceability for dependency identification among models. To enable a unified treatment of all models and related data, we integrated additional metamodels into EMFStore to maintain and store traceability links and rules directly in the repository. Our traceability metamodel provides explicit support for modeling a hierarchy of dependency types to ensure the proper categorization of detected dependencies. Dependency types can be created and maintained by users in the repository and aggregated in special catalogs, and are therefore subject of model versioning as well. The metamodel for traceability rules offers a similar concept, which enables users to create rules directly in the repository and group them together in catalogs, to support fast exchange and easy maintenance. The design of our rules is based on similar approaches as proposed in [3] and [4]. However, we enhanced our rules with information retrieval techniques which supply additional query-operators to improve dependency detection. More details on the concepts for dependencies and rules follow in the next subsections.

1) Dependency Concept: We store dependency information as traceability links to benefit from their rich semantics. Our traceability metamodel supports binary and n-ary traceability links. Each traceability link can be enhanced with additional information such as the type of a dependency, design decisions, design alternatives, and

[1] http://modelrepository.sourceforge.net/project-summary.html
[2] http://wiki.eclipse.org/CDO
[3] http://www.eclipse.org/proposals/emf-store/
[4] http://www.andromda.org/index.php
[5] http://www.eclipse.org/modeling/emf/

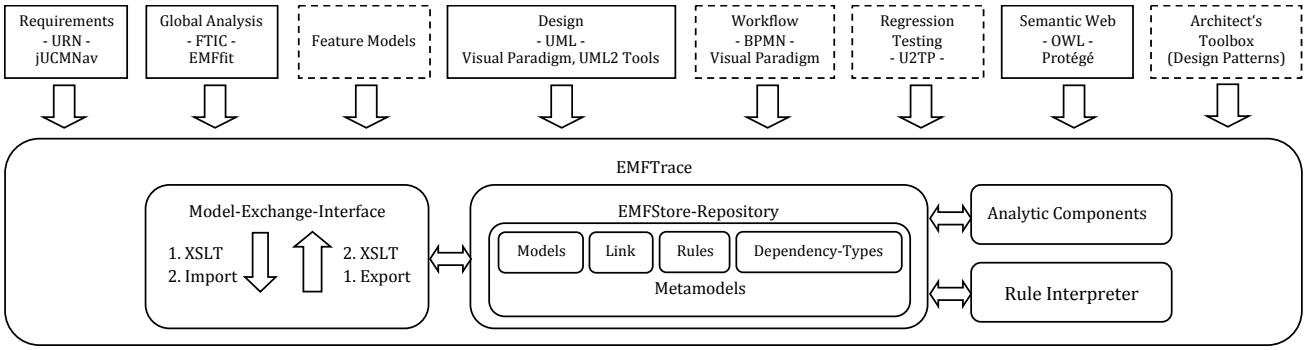


Figure 1. Conceptual overview of EMFTrace and its central role as a platform for various coupled research activities. The integration of dashed items is subject of current research and development.

assessments of the confidence of the link performed by users.

Moreover, we are able to combine transitively related traceability links, i.e., links that share common model elements as their source or target elements, into traces. These traces store chains of links to span design decisions and multi-level dependency relations. The example shown in Figure 2 illustrates the transition from a URN goal to its architectural realization through UML models. Furthermore, EMFTrace provides automated validation features for traceability links and traces, to ensure the integrity of stored dependency information. Corrupt or outdated links will be erased while broken traces are either split into smaller sub-traces or removed as well, if they contain less than two links.

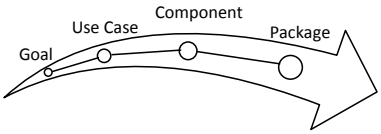


Figure 2. A trace containing a transitive chain of links

2) Rule Concept: Following the general design proposed in [3] and [4], our rules comprise three parts as shown in Figure 3, whereas each part is responsible for a certain task.

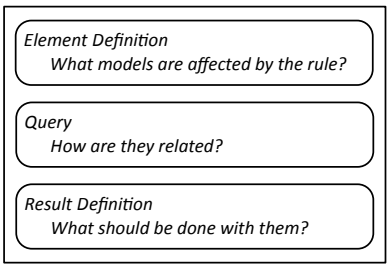


Figure 3. Main components of our rules

In contrast to existing approaches as [2], we do not operate on files, but on the conceptual level of models inside the EMFStore repository. Each rule can be equipped with several actions carried out once the conditions are met. This enables us to create a link for a relationship and its inverse relationship at once, without requiring additional rules. Rules can be created and edited by users in the repository using EMF-based editors or external in their XML representation, and can be imported afterwards. Rules can be grouped in catalogs to enable the use of specialized rule catalogs for different tasks such as dependency detection or consistency validation, to improve the useability of our tool. Currently there is one catalog available which contains 68 rules for traceability detection. Our rules operate directly on model attributes and the structure of models. They compare attributes to identify dependencies between models, for example the name attribute of an OWL class with the name attribute of an UML component. But since the comparison for exact matching is not always sufficient to detect all dependencies, information retrieval techniques, such as n-gram based string comparison, are used to compute the similarity of attributes.

B. Architecture and Realization

1) Architecture: Since EMFTrace is our evolving basis for many different research activities related to model-driven engineering, its architecure must support the addition of new concepts and features. Our entire tool is based on Eclipse technology and consists of several plug-ins that provide the core functionality, a user interface, and our adapted metamodels. EMFStore provides a server (the repository) and a client to operate with the repository, which is why we integrated our additional features into the client to provide a unified view and usability. We designed several additional components that integrate new features into the EMFStore client, such as an interpreter for traceability rules as shown in the bottom right corner of Figure 1. Our architecture is extensible and allows for easy integration of new components into EMFTrace to introduce new features, i.e., new analysis components. All plug-ins and components were implemented and tested with Java to ensure the platform independent applicability of our tool [10].

2) Model Integration: Using a modeling language with EMFTrace requires the integration of its metamodel into EMFStore. The metamodel must be converted into an

EMF-based Ecore model, which can either be modeled with EMF or generated from an XSD file. Once an Ecore model has been generated, it must be adapted to EMFStore's metamodel, which provides additional attributes to facilitate the management through EMFStore. We performed these steps for the metamodels of UML, URN, and OWL.

Since the adaptation to EMFStore's metamodel introduces new attributes to the metamodels, an adaptation of models (metamodel instances) is required as well, to add these attributes while importing models from CASE tools. We perform the integration and adaptation as illustrated by the arrows in the middle of Figure 1 by applying XSLT templates on models and encapsulated the entire import and export procedures in new components (*Model-Exchange-Interface*), which have been integrated into EMFTrace. Therefore, XML-based models exported by CASE tools and XSLT provide the core technology for our integration approach.

As CASE tools (see top of Figure 1) we currently support the Eclipse UML2Tools (UML), Visual Paradigm (UML, BPMN), the Eclipse-based jUCMnav (URN), and Protégé (OWL) with our XSLT templates. Additional modeling languages and their respective CASE tools such as BPEL can be added to EMFTrace at any time through adding an EMF-based metamodel to the underlying EMFStore repository and providing further XSLT templates.

3) Extensions: One addition which has already been integrated into EMFTrace is the tool EMFfit [12] (see Figure 1, top left), which supports the management of factor tables and issue cards for Global Analysis as proposed by Hofmeister et al. [11] and therefore contributes a new metamodel to EMFTrace. The tool itself is implemented and tested as a set of Eclipse plug-ins with Java and offers support for the transition from requirements to architectural design.

IV. Conclusion and Further Work

We presented an approach and a prototype tool EMFTrace for model integration and dependency identification between models of the entire software development process. We support different artifacts ranging from requirements models to design models which are managed by a unifying repository. Our tool provides comprehensive and automated dependency identification through rule-based traceability to ensure decent precision of results, enhanced with information retrieval techniques. EMFTrace facilitates the use of semantically important link types and is capable of storing, versioning, and maintaining traceability links. Standard modeling languages, such as UML, URN, and OWL, are integrated into the repository and several CASE tools are supported.

Further work will focus on several issues: additional modeling languages such as U2TP and feature models shall be integrated into EMFTrace along with the appropriate metamodels and XSLT templates, allowing for additional CASE tools to be used in conjunction with our tool. New models enable new analysis capabilities and require new rules or existing rules to be refined. Our rule concept should be enhanced to support consistency checks of models stored in the repository. The combination of dependency identification with consistency analysis provides the basic features to perform change impact analysis on models through new components and rules. To improve the usability of our tool, model synchronization with CASE tools and methods of change recognition for maintenance are considered as further enhancements. Drawing more benefits from discovered dependency relations requires sophisticated visualization of thereby created traceability links to enable the user to navigate on hierarchical chains of dependencies and to trace models efficiently. Finding appropriate means of dependency visualization is therefore one important research question related to EMFTrace.

References

[1] G. A. A. C. Filho, A. Zisman, and G. Spanoudakis, "Traceability approach for i* and UML models," in *Proceedings of 2nd International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS'03)*, 2003.

[2] W. Jirapanthong and A. Zisman, "Xtraque: traceability for product line systems," *Software and Systems Modeling*, vol. 8, no. 1, pp. 117–144, 2009.

[3] G. Spanoudakis, A. d'Avila Garces, and A. Zisman, "Revising rules to capture requirements traceability relations: A machine learning approach," in *Proc. Int. Conf. in Software Engineering and Knowledge Engineering (SEKE 2003)*. Knowledge Systems Institute, Skokie, 2003, pp. 570–577.

[4] G. Spanoudakis, A. Zisman, E. Perez-Minana, and P. Krause, "Rule-based generation of requirements traceability relations," *JSS*, vol. 72, no. 2, pp. 105–127, 2004.

[5] A. D. Lucia, F. Fasano, R. Oliveto, and G. Tortora, "Recovering traceability links in software artifact management systems using information retrieval methods," *ACM Trans. Softw. Eng. Methodol.*, vol. 16, no. 4, Article 13, Sept. 2007.

[6] M. Koegel and J. Helming, "EMFStore: a model repository for EMF models," in *Proc. Int. Conf. on Software Engineering (ICSE'10)*. ACM, 2010, pp. 307–308.

[7] ITU-T, "Recommendation ITU-T Z.151 User requirements notation (URN) – Language definition," ITU-T, Nov 2008.

[8] S. Winkler and J. von Pilgrim, "A survey of traceability in requirements engineering and model-driven development," *Softw. and Syst. Model.*, vol. 9, no. 4, pp. 529–565, 2010.

[9] A. Marcus and J. I. Maletic, "Recovering documentation-to-source-code traceability links using latent semantic indexing," in *Proc. Int. Conf. on Software Engineering (ICSE'03)*. IEEE, 2003, pp. 125–135.

[10] S. Lehnert, "Softwarearchitectural Design and Realization of a Repository for Comprehensive Model Traceability," Diploma thesis, Ilmenau University of Technology, Ilmenau, Germany, November 2010.

[11] C. Hofmeister, R. Nord, and D. Soni, *Applied Software Architecture*. Boston, MA, USA: Addison-Wesley, 2000.

[12] P. Wagner, "Tool Support for the Analysis during Software Architectural Design," Bachelor thesis, Ilmenau University of Technology, Ilmenau, Germany, December 2010.

Combining Multiple Dimensions of Knowledge in API Migration

Thiago Tonelli Bartolomei[1], Mahdi Derakhshanmanesh[2], Andreas Fuhr[2], Peter Koch[2], Mathias Konrath[2], Ralf Lämmel[2], and Heiko Winnebeck[2]
[1] *University of Waterloo, Canada*
[2] *University of Koblenz-Landau, Germany*

***Abstract*—We combine multiple dimensions of knowledge about APIs so that we can support API migration by wrapping or transformation in new ways. That is, we assess wrapper-based API re-implementations and provide guidance for migrating API methods. We demonstrate our approach with two major GUI APIs for the Java platform and two wrapper-based re-implementations for migrating between the GUI APIs.**

***Keywords*-Software migration, API migration, API analysis, Wrapping, Mining software repositories**

I. Introduction

API migration is a kind of software migration; it may be necessary to meet requirements for software modernization, application integration, and others. API migration is realized by wrapping or transformation. We refer to [1], [2], [3], [4], [5], [6], [7], [8] for recent work on the subject.

For instance, consider the following re-engineering scenario. Two Java applications need to be integrated, but they use different GUI APIs, say SWING and SWT. Based on the exercised features and possibly other considerations, one of the two APIs is favored for the integrated application. The disfavored API (the "source API") can be re-implemented in terms of the favored API (the "target API") as a *wrapper* so that the migration requires little, if any, rewriting of the application's code. Incidentally, there are two advanced open-source wrappers that serve both directions of migration: SWINGWT[1] and SWTSWING[2].

In previous work [6], [8], we substantiated that migration between independently developed source and target APIs may be complex because of significantly different generalization hierarchies, contracts, and protocols.

Contribution: In the present paper, we describe an approach for the combination of multiple dimensions of knowledge about APIs so that API migration can be supported in new ways. That is, we assess wrapper-based API re-implementations and provide guidance for migrating API methods. To this end, we leverage a model-based approach to the integration of knowledge about APIs into a repository for convenient use in declarative queries. Throughout the paper, we use the SWING/SWT APIs and the above-mentioned wrappers as subjects under study.

Road-map: Sec. II describes the integrated repository. Sec. III and Sec. IV cover different forms of supporting API migration. Related work is discussed in Sec. V, and the paper is concluded in Sec. VI. The paper and accompanying material are available online.[3]

[1] http://swingwt.sourceforge.net/: re-implements SWING in terms of SWT
[2] http://swtswing.sourceforge.net/: re-implements SWT in terms of SWING
[3] http://softlang.uni-koblenz.de/apirep/

Acknowledgement We are grateful to Daniel Ratiu for providing us with data related to the programming ontology of [9], [10]. We are also grateful to four anonymous MDSM 2011 reviewers for their excellent advice.

II. The integrated repository

We integrate three data sources with API knowledge into a repository. Let us describe those data sources, the metamodel of the integrated repository, and the repository technology as such.

A. Data sources

- **APIMODEL** (developed by the present authors)—a model of API implementations (including SWING, SWT, SWINGWT, SWTSWING) with an underlying metamodel that is a (very) limited Java metamodel for structural properties and calling relationships;
- **APIUSAGE** (developed by Lämmel et al. [11])—a fact base (say, database) with usage properties of 1476 open-source Java projects at SourceForge, in particular with facts for API method calls within the projects' code;
- **APILINKS** (developed by Ratiu et al. [9], [10])—an ontology for programming concepts that were extracted semi-automatically from APIs in different programming domains, complete with trace links between concepts and the API source-code elements from which they were derived.

The APIMODEL source contributes basic knowledge about types and methods of genuine API implementations, and their coverage by the typically incomplete wrapper-based re-implementations. The APIUSAGE source helps to assess, for example, the relevance of genuine methods that are not implemented in a wrapper. The APILINKS source helps to derive candidate classes and methods that could be used in a wrapper-based API re-implementation.

B. Metamodel of the repository

Fig. 1 shows the metamodel (a UML class diagram) of our integrated repository where metaclasses are tagged by data sources APIMODEL, APIUSAGE, and APILINKS. We must note that the metamodel does not cover all elements of the sources, but is streamlined to fit our objectives.

The metaclass *NamedElement* represents package-qualified names of packages, classes, and methods. Because of the composition relationships in the metamodel, *NamedElement*s are also qualified by the name of an API, in fact, by a particular implementation, which could be a genuine implementation or a wrapper-based re-implementation.

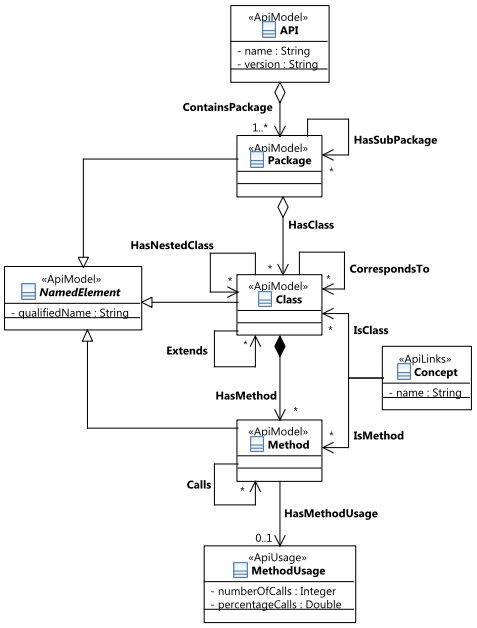


Figure 1. Metamodel of the integrated repository with API knowledge

The metaclasses *Package*, *Class*, and *Method* represent the package hierarchy with the Java classes and their methods, further with extension relationships between classes (see association *Extends*) and calling relationships between methods (see association *Calls*). As a means of prioritization, we leave out interfaces; they are trivially copied by wrappers.

Classes of genuine API implementations are linked with the corresponding classes of wrappers (see association *CorrespondsTo*). Here we note that wrappers may use different package prefixes. Also, these links improve convenience for those queries that need to navigate between the different API implementations. The metaclass *Concept* models concepts in the sense of APILINKS' ontology. Classes and methods can be linked with concepts; see associations *IsClass* and *IsMethod*. Hence, classes and methods of different APIs may be linked transitively.

The metaclass *MethodUsage* represents the usage data that was integrated from APIUSAGE. That is, for each API method, we maintain the *number of calls* to the method (if any) within the SourceForge projects covered by APIUSAGE [11]. We translated this number also into a relative measure in the sense of the *percentage of the calls* to the given method relative to the number of all calls to methods of the API.

C. Repository technology

The repository leverages the model-based TGraph approach [12]. The metamodel of Fig. 1 is represented as a TGraph schema; converters instantiate the schema from the different data sources. All analysis is performed by means of queries on TGraphs using the language *GReQL* (Graph Repository Query Language) [13]. For brevity, we describe all queries ("measurements") only informally in this paper, but here is a simple, illustrative GReQL example for retrieving all classes c of an API a that are not implemented by a wrapper:

```
using a:
from c: V{Class}
with c.qualifiedName =~ a and count(c-->{CorrespondsTo}) = 0
reportSet c
end
```

That is, a is an argument of the query for the name of the API; the query selects ("reports") all classes c such that the qualified name of c matches with a and there are no outgoing edges of the type *CorrespondsTo* (see `-->{CorrespondsTo}`) from c.

III. WRAPPER ASSESSMENT

Consider again our introductory scenario for API migration. Which wrapper, SWINGWT or SWTSWING, should we favor? Such decision making should take into account wrapper qualities, e.g., its completeness or compliance—both relative to the genuine API implementation. In case we want to improve a given wrapper, we should also track progress by simple metrics. Accordingly, we propose some concepts for wrapper assessment.

A. Coverage of source API

We can trivially compare the APIMODEL data between genuine API implementation and wrapper to get a basic sense of completeness in terms of (the percentage of) genuine packages, classes, and methods that are covered (say, re-implemented) by the wrapper. Table I collects such metrics for the SWING/SWT wrappers. The numbers show that the wrappers are highly incomplete.

	SWINGWT	SWTSWING
Packages	25 (78.12 %)	16 (51.61 %)
Classes	533 (18.61 %)	372 (56.97 %)
Methods	4533 (26.60 %)	3426 (42.59 %)

Table I
COVERAGE OF SOURCE API

B. Wrapper compliance issues

Some forms of non-compliance of a wrapper with the genuine API implementation can be determined by simple queries on our repository, e.g., differences regarding generalization hierarchies or the declaring classes for methods. Consider the following extension chain for SWING's *AbstractButton*:

```
java.lang.Object
 |_ java.awt.Component
     |_ java.awt.Container
         |_ javax.swing.JComponent
             |_ javax.swing.AbstractButton
```

The chain itself is preserved by SWINGWT. However, SWING declares the method *addActionListener* on the class *AbstractButton* whereas SWINGWT declares the method already on the class *Component*.

	SWINGWT	SWTSWING
• Declarations on supertypes	516	161
• Empty implementations	1006	230
• Missing methods	12506	4618
○ Class missing	9604	3698
○ Class present	2902	920

Table II
WRAPPER COMPLIANCE ISSUES

Table II shows numbers for some metrics for (lack of) wrapper compliance. In reference to the above example of the method *addActionListener*, we measure the number of methods that are declared "earlier" on a supertype in the wrapper. Further, we measure methods with empty implementations, i.e., implementations without any outgoing method calls, while the corresponding genuine implementations had outgoing method calls. (The substantial number of empty implementations may be surprising, but these wrappers are nevertheless reportedly useful in practice.) Finally, we also subdivide missing methods into those that are implied by missing classes vs. those that are missing from existing classes.

C. Relevance in terms of usage

Let us qualify wrapper (in-) completeness with APIUSAGE data. If the developers of the wrappers applied the right judgement call for leaving out classes and methods, then the missing methods should be less relevant in practice than the implemented ones. Table III lists usage metrics for the SWING/SWT wrappers.

	SWINGWT	SWTSWING
Unimplemented methods		
• Any usage	9,01 %	2,90 %
• Cumulative usage	2,88 %	2,35 %
Empty methods		
• Any usage	42,53 %	25,71 %
• Cumulative usage	11,41 %	1,49 %
Non-empty methods		
• Any usage	48,46 %	71,39 %
• Cumulative usage	85,72 %	96,17 %

Table III
USAGE OF API METHODS IN SOURCEFORGE

In the table, we break down SWING's and SWT's methods into categories according to the wrappers as follows: unimplemented, empty, and non-empty implemented methods. For each category, we show the percentage of methods with "any usage" (say, any calls) in the SourceForge projects in the scope of the APIUSAGE source. We also show "cumulative usage" for each category, i.e., the contribution of the category to all API method calls. These are contrasting numbers which show, for example, that the many unimplemented and empty methods (see again Table II) are exercised much less frequently than the fewer non-empty methods.

IV. GUIDANCE FOR MIGRATION

A given wrapper may be effectively incomplete in that a missing method is actually exercised by the application under API migration. In this case, we seek guidance for migrating the API method in question. Such guidance is universally useful for API migration—even when transformation is used instead of wrapping. A practical approach to guidance would need to combine elements of API type matching, IDE support (such as autocompletion and stub generation), and others. We focus here on the aspect of proposing method candidates to be called in methods of wrapper-based API re-implementations.

A. Concept-based method candidates

We can use APILINKS' trace links between API methods and concepts to propose method candidates. The idea is that if methods of the source and target APIs are related to the same concept, then the latter may be useful in re-implementing the former. Further, let us sort all such candidates by their cumulative usage, say, by their relevance as far as APIUSAGE is concerned.

Qualified candidate name	Cumulative usage (%)
swing.javax.swing.ImageIcon.ImageIcon	**0,4816**
swing.java.awt.image.BufferedImage.BufferedImage	0,1063
swing.java.awt.Frame.getIconImage	0,0059
swing.java.awt.....MemoryImageSource	0,0046
swing.java.awt.Frame.setIconImage	0,0042
swing.javax.swing.text.html.ImageView.ImageView	0,0005
swing.java.awt.....ImageGraphicAttribute	N/A

Table IV
CANDIDATES FOR RE-IMPLEMENTING SWT'S *Button.setImage*

Suppose you need to migrate SWT's *Button.setImage* to SWING. Table IV shows the method candidates that were automatically determined by a GReQL query. Consider the first line with the constructor of *ImageIcon*. We show the line in bold face to convey the fact that there is an existing wrapper, SWTSWING, whose method implementation of *setImage* readily involves the constructor of *ImageIcon*.

Further inspection reveals that SWING's *JButton*, which is a counterpart to SWT's *Button*, does not provide an *Image* property and, hence, we cannot simply migrate SWT's *Button.setImage* to a corresponding setter of SWING. Extra state and a more complex idiom (indeed involving *ImageIcon*) is needed.

B. Assessment of the ontology

The above example shows that APILINKS may suggest reasonable candidates—in principle. We would like to assess APILINKS's relevance more generally. In particular, we could compare APILINKS-based links with actual calling relationships in existing wrapper implementations, as they are available through APIMODEL's data. Table V lists corresponding metrics for the SWING/SWT wrappers.

	SWINGWT	SWTSWING
Unimplemented methods with links	10.83 %	0.35 %
Implemented methods with links	28.06 %	24.98 %
Correct links	42.75 %	37.20 %

Table V
API LINKS BETWEEN SWING AND SWT

The coverage of API parts by APILINKS' trace links is an artifact of the underlying semi-automatic ontology extraction approach [9], [10], which involves elements of name matching and thresholds for the inclusion of concepts. We cannot expect to retrieve links for arbitrary methods from APILINKS.

In the table, we break down SWING's and SWT's methods into the categories of unimplemented and implemented methods according to the wrappers. For both categories, we show the percentage of methods that are linked (transitively) with one or more methods of the

corresponding target API. The numbers are such that implemented methods happen to be much better linked than unimplemented ones.

At the bottom of the table, we also list the percentage of correct APILINKS' trace links. We say that a link from the method m of the source API s to a method m' of the target API t is correct, if a given wrapper-based re-implementation of s in terms of t implements m in a way that it directly calls m'. When we specify the percentage, we consider as the baseline (100%) only those methods m that both have associated trace links to t and actually call some method of t. It turns out that APILINKS predicts a correct link in more than 1/3 of the cases. We have to note though that APILINKS typically proposes multiple candidates—with a median of 8.

V. Related work

Work on API migration has previously focused on transformation and wrapper-generation techniques for API upgrades [2], [3], [4], [5] and, to a lesser extent, on migration between independently developed APIs [1], [6], [7], [8]. The present work is the first to integrate diverse data sources to assess wrappers and to guide their development. Typically, wrappers are assessed by *testing* (i.e., testing whether the application under migration continues to function, or recovers from any test failures that had to be addressed by improving a pre-existing wrapper) [6]. There is no previous work on guiding API-wrapper development for independently developed APIs.

Most of the techniques that we integrate are inspired by program comprehension research. For instance, our comparison of different API implementations is a simple form of object-model matching [14]. Also, our exploitation of API-usage data is straightforward, when compared to other scenarios of exploiting such data in the context of API usability [15] and understanding API usage (patterns) [16], [17]. Our proposal for guided migration can be viewed as one specific approach to advanced ("intelligent") code completion systems [18], [19].

VI. Concluding remarks

The complexity of API migration requires many skills and techniques. Of course, one must understand the API's domain, and the application under migration. Basic software engineering skills such as testing, design by contract, effective use of documentation are critical as well. Still API migrations are largely unstructured today, and they come with unpredictable costs. We submit that techniques for assessment and guidance, such as those discussed in this short paper, are needed to tackle non-trivial API migrations in the future.

Clearly, our work is at an early state, and makes only a limited contribution to the larger API migration theme. There is a need for a comprehensive approach for guided API migration, which should combine diverse elements of assessment, mapping, matching, code completion, code generation, and testing.

References

[1] I. Balaban, F. Tip, and R. Fuhrer, "Refactoring support for class library migration," in *Proc. of OOSPLA 2005*. ACM, 2005, pp. 265–279.

[2] J. Henkel and A. Diwan, "CatchUp!: capturing and replaying refactorings to support API evolution," in *Proc. of ICSE 2005*. ACM, 2005, pp. 274–283.

[3] J. H. Perkins, "Automatically generating refactorings to support API evolution," in *Proc. of the Workshop on Program Analysis for Software Tools and Engineering (PASTE)*. ACM, 2005, pp. 111–114.

[4] I. Şavga, M. Rudolf, S. Götz, and U. Aßmann, "Practical refactoring-based framework upgrade," in *Proc. of the Conference on Generative Programming and Component Engineering (GPCE)*. ACM, 2008, pp. 171–180.

[5] D. Dig, S. Negara, V. Mohindra, and R. Johnson, "ReBA: refactoring-aware binary adaptation of evolving libraries," in *Proc. of ICSE 2008*. ACM, 2008, pp. 441–450.

[6] T. T. Bartolomei, K. Czarnecki, R. Lämmel, and T. van der Storm, "Study of an API Migration for Two XML APIs," in *Proc. of Conference on Software Language Engineering (SLE 2009)*, ser. LNCS, vol. 5969. Springer, 2010, pp. 42–61.

[7] M. Nita and D. Notkin, "Using Twinning to Adapt Programs to Alternative APIs," in *Proc. of ICSE 2010*, 2010.

[8] T. T. Bartolomei, K. Czarnecki, and R. Lämmel, "Swing to SWT and Back: Patterns for API Migration by Wrapping," in *Proc. of ICSM 2010*. IEEE, 2010, 10 pages.

[9] D. Ratiu, M. Feilkas, and J. Jürjens, "Extracting Domain Ontologies from Domain Specific APIs," in *12th European Conference on Software Maintenance and Reengineering, CSMR 2008, Proceedings*. IEEE, 2008, pp. 203–212.

[10] D. Ratiu, M. Feilkas, F. Deissenboeck, J. Jürjens, and R. Marinescu, "Towards a Repository of Common Programming Technologies Knowledge," in *Proc. of the Int. Workshop on Semantic Technologies in System Maintenance (STSM)*, 2008.

[11] R. Lämmel, E. Pek, and J. Starek, "Large-scale, AST-based API-usage analysis of open-source Java projects," in *SAC'11 - ACM 2011 SYMPOSIUM ON APPLIED COMPUTING, Technical Track on "Programming Languages"*, 2011, to appear.

[12] J. Ebert, V. Riediger, and A. Winter, "Graph Technology in Reverse Engineering: The TGraph Approach," in *WSR 2008*, ser. GI-EditionProceedings, vol. 126. Gesellschaft für Informatik, 2008, pp. 67–81.

[13] D. Bildhauer and J. Ebert, "Querying Software Abstraction Graphs," in *Query Technologies and Applications for Program Comprehension (QTAPC 2008), Workshop at ICPC 2008*, 2008.

[14] Z. Xing and E. Stroulia, "UMLDiff: an algorithm for object-oriented design differencing," in *20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005), Proceedings*. ACM, 2005, pp. 54–65.

[15] J. Stylos, B. A. Myers, and Z. Yang, "Jadeite: improving API documentation using usage information," in *Proc. of the 27th Intern. Conf. on Human Factors in Computing Systems, CHI 2009*. ACM, 2009, pp. 4429–4434.

[16] J. Stylos and B. A. Myers, "Mica: A Web-Search Tool for Finding API Components and Examples," in *2006 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2006), Proceedings*. IEEE, 2006, pp. 195–202.

[17] T. Xie and J. Pei, "MAPO: mining API usages from open source repositories," in *MSR '06: Proceedings of the 2006 international workshop on Mining software repositories*. ACM, 2006, pp. 54–57.

[18] D. Mandelin, L. Xu, R. Bodík, and D. Kimelman, "Jungloid mining: helping to navigate the API jungle," in *Proc. of the 2005 ACM SIGPLAN conference on Programming language design and implementation (PLDI 2005)*. ACM, 2005, pp. 48–61.

[19] M. Bruch, M. Monperrus, and M. Mezini, "Learning from examples to improve code completion systems," in *Proceedings of ESEC/SIGSOFT FSE 2009*. ACM, 2009, pp. 213–222.

Proceedings of the

Fifth International Workshop on Software Quality and Maintainability (SQM 2011)

Fifth International Workshop on Software Quality and Maintainability

Bridging the gap between end-user expectations, vendors' business prospects, and software engineers' requirements on the ground.

Magiel Bruntink
Software Improvement Group
The Netherlands
m.bruntink@sig.eu

Kostas Kontogiannis
School of Electrical and Computer Engineering
National Technical University of Athens
Greece
kkontog@softlab.ece.ntua.gr

Preface

The fifth international workshop on Software Quality and Maintainability (SQM 2011) offered a forum to researchers to present their original work and to practitioners to relate their experiences on issues pertaining to software quality and maintainability. Moreover, the theme of the workshop invited discussion on how to bridge the gap between end user expectations, business requirements, vendor performance, and engineering constraints regarding software quality.

SQM 2011 was held as a satellite event of the 15th European Conference on Software Maintenance and Reengineering (CSMR 2011). In 2010, the fourth SQM workshop was held as a satellite event of CSMR 2010.

Carl Worms of Credit Suisse kicked-off the workshop with an invited talk titled "Software Quality Management - Quo Vadis?" A short paper describing his current work is included in these proceedings.

In this volume, you will further find the papers accepted for presentation at the workshop. Out of 8 full-paper submissions, 6 papers were selected. The accepted papers were published at CEUR-WS[1].

Theme & Goals

Software is playing a crucial role in modern societies. Not only do people rely on it for their daily operations or business, but for their lives as well. For this reason, correct and consistent behaviour of software systems is a fundamental part of end user expectations. Additionally, businesses require cost-effective production, maintenance, and operation of their systems. Thus, the demand for good quality software is increasing and is setting it as a differentiator for the success or failure of a software product. In fact, high quality software is becoming not just a competitive advantage but a necessary factor for companies to be successful.

The main question that arises now is how quality is measured. What, where and when we assess and assure quality, are still open issues. Many views have been expressed about software quality attributes, including maintainability, evolvability, portability, robustness, reliability, usability, and efficiency. These have been formulated in standards such as ISO/IEC-9126 and CMMI. However, the debate about quality and maintainability between software producers, vendors and users is ongoing, while organizations need the ability to evaluate the software systems that they use or develop from multiple angles.

So, is "Software quality in the eye of the beholder"? This workshop session aims at feeding into this debate by establishing what the state of the practice and the way forward is.

Accepted papers

- *Automated Quality Defect Detection in Software Development Documents*, Andreas Dautovic, Reinhold Ploesch and Matthias Saft.
- *Design Pattern Detection using Software Metrics and Machine Learning*, Satoru Uchiyama, Atsuto Kubo, Hironori Washizaki and Yoshiaki Fukazawa.
- *Using the Tropos Methodology to Increase the Quality of Software Design*, Andrea Capiluppi and Cornelia Boldyreff.
- *Tool-Supported Estimation of Software Evolution Effort in Service-Oriented Systems*, Johannes Stammel and Mircea Trifu.
- *Preparing for a Literature Survey of Software Architecture using Formal Concept Analysis*, Luís Couto,

[1] http://ceur-ws.org

Miguel Alexandre Ferreira, Eric Bouwers and José Nuno Oliveira.

- *Evidence for the Pareto principle in Open Source Software Activity*, Mathieu Goeminne and Tom Mens.

Organization

Chairs

- Magiel Bruntink, Software Improvement Group, The Netherlands
- Kostas Kontogiannis, National Technical University of Athens, Greece
- Miguel Alexandre Ferreira (publicity chair), Software Improvement Group, The Netherlands

Program Committee

- Árpád Beszédes, University of Szeged, Hungary
- Andrea De Lucia, University of Salerno, Italy
- Florian Deissenboeck, Technische Universität München, Germany
- Massimiliano Di Penta, University of Sannio, Italy
- Juergen Ebert, University of Koblenz-Landau, Germany
- Slinger Jansen, Utrecht University, the Netherlands
- Rainer Koschke, University of Bremen, Germany
- Robert Lagerström, the Royal Institute of Technology, Sweden
- Radu Marinescu, Politehnica University of Timisoara, Romania
- Liam O'Brien, National ICT Australia Limited, Australia
- Ladan Tahvildari, University of Waterloo, Canada
- Arie van Deursen, Delft University of Technology, the Netherlands
- Jurgen Vinju, Centrum Wiskunde & Informatica, the Netherlands
- Joost Visser, Software Improvement Group, the Netherlands
- Carl Worms, Credit Suisse, Switzerland
- Hongyu Zhang, Tsinghua University, China
- Christos Tjortjis, University of Ioannina & University of Western Macedonia, Greece
- Alexander Chatzigeorgiou, University of Macedonia, Greece
- Jesus M. Gonzalez-Barahona, Universidad Rey Juan Carlos, Spain
- Rudolf Ferenc, University of Szeged, Hungary

Sub-reviewers

- Markus Buschle
- Péter Hegedüs
- Giuseppe Scanniello
- Gabriella Tóth

Sponsors

Software Improvement Group
Amsterdam, The Netherlands

Acknowledgements

We are grateful to all members of the Program Committee and to their sub-reviewers for helping to make SQM 2011 a success. Many thanks to Carl Worms for his keynote talk. Also we would like to thank the Software Improvement Group for sponsoring and hosting our website, and EasyChair.org for their invaluable conference organization tool.

February 2011,
Magiel Bruntink and Kostas Kontogiannis
Chairs SQM 2011

Invited Keynote: Software Quality Management – quo vadis?

Carl Worms
Quality and Process Management
Credit Suisse AG
Zürich, Switzerland
carl.f.worms@credit-suisse.com

Software engineering as a discipline startet end of the sixties as answer to the "software crisis" – the term "software quality" wasn't even used. Since then, it took nearly additional 30 years to establish a normative software quality management with ISO 9000-3 in 1997, the particular "interpretation" of the general ISO 9000 from 1994 for software development organizations. At the same time, often driven by governments and military as largest IT sponsors, procedure models like the German V-Modell '97, the spriral model and somewhat later the Rational Unified Process have been developed, accompanied by assessment models like CMMI or SPICE. Nowadays, there's no successful software company on the market without defined procedure model and/or a CMMI or SPICE maturity level 3, and suppliers for governments or big companies are only accepted for bidding if they are certified against one of these standards.

Though, there seems to be still no significant change in the yearly Chaos reports, there still dramatically fail software projects – or even running software - what's wrong, the sensation or software quality in reality?

The talk comprises the last 20 years of software quality management from the view of a software engineer and quality manager in medium and big software development organizations, comparing the hidden impact of changing organizational behaviour and structures on software architecture and quality – and vice versa.

Facts and hypotheses on the following topics get discussed:

Get requirements constrained by natural laws better implemented than those constrained by social contracts (i.e. business processes)?

Is there any empirically proven impact of the number of transformation steps (between informal requests and requirements in natural language to the final compilation into hardware language) on software quality – and its cost?

Which promising areas for software engineering research can be seen in the realm of very large IT systems?

For the latter, practical examples and experience from the biggest IT hub of a global bank is shown for topics like

- evolution of a large application landscape
- organizational impact on software architecture
- the way from spaghetti to tagliatelle to letter soup and: what's the soup?
- pitfalls in global distributed software development and operations
- the common ground of quality management, business analysis and enterprise architecture.

The talk concludes with observed gaps in the education curricula for software engineers and managers.

Carl Worms is enterprise architect in Credit Suisse IT Private Banking with focus on strategy and governance of solution delivery processes. Since 1991, he worked in several large enterprises in the areas of software engineering methodology and software quality management. In 1993 he got the Walter Masing Award of the German Society for Quality with a paper on object orientation and automated testing. He joined Credit Suisse IT architecture in 1999 as leading methodologist and led the first software process improvement program from 2002-2005. In 2006-2008 he was head of the IT Private Banking Quality Management unit and since 2008 is lead architect of this unit. (In November 2010 Credit Suisse IT Private Banking/Region Switzerland became one of the largest IT organizations outside India on CMMI DEV maturity level 3.)

Automated Quality Defect Detection in Software Development Documents

Andreas Dautovic, Reinhold Plösch
Institute for Business Informatics - Software Engineering
Johannes Kepler University Linz
Altenberger Straße 69, 4040 Linz, Austria
andreas.dautovic | reinhold.ploesch@jku.at

Matthias Saft
Corporate Technology
Siemens AG
Otto-Hahn-Ring 6, 81739 Munich, Germany
matthias.saft@siemens.com

***Abstract*—Quality of software products typically has to be assured throughout the entire software development life-cycle. However, software development documents (e.g. requirements specifications, design documents, test plans) are often not as rigorously reviewed as source code, although their quality has a major impact on the quality of the evolving software product. Due to the narrative nature of these documents, more formal approaches beyond software inspections are difficult to establish. This paper presents a tool-based approach that supports the software inspection process in order to determine defects of generally accepted documentation best practices in software development documents. By means of an empirical study we show, how this tool-based approach helps accelerating inspection tasks and facilitates gathering information on the quality of the inspected documents.**

Keywords-quality defect detection; software development document; tool-based approach; software inspection

I. INTRODUCTION

Software quality assurance aims at ensuring explicitly or implicitly defined quality goals for a software product. Assuring the quality of a software product basically deals with the fulfillment of specified functional and quality requirements, where the checks are often realized by static and dynamic testing of the software product. Software development documents like requirements specifications, which define how to build the right software product, or design documents, which define how to build the software product right, are also an essential part of the entire software product. However, they are often not treated with the same enthusiasm as source code. Consequently, software bugs are fixed in a later phase of the software product life-cycle, which leads to increased costs for software changes [1]. For instance, a cost/benefit-model reveals that due to the introduction of design inspection 44 percent of defect costs compared to testing alone can be saved [2]. Therefore, to positively influence the development of a software product, quality assurance also has to systematically deal with the quality of software development documents.

Natural language is a commonly used representation for software development documents. However, as a result of its informal nature, natural language text can easily lead to inadequate or poor project documentation, which makes software hard to understand, change or modify. Based on a comprehensive literature study, Chen and Huang [3] identified five quality problems of software documentation:

- Documentation is obscure or untrustworthy.
- System documentation is inadequate, incomplete or does not exist.
- Documentation lacks traceability, as it is difficult to trace back to design specifications and user requirements.
- Changes are not adequately documented.
- Documentation lacks integrity and consistency.

In order to improve the overall quality of natural language project documents throughout the software life-cycle, the use of inspections is generally accepted. Since the introduction of inspections in the mid-1970s by Fagan [4], some modifications have been made to the original process. Improved reading techniques including: checklist-based reading [5] [6], usage-based reading [6] [7] or perspective-based reading [8] [9] are nowadays available for checking consistency and completeness of natural language texts. However, analysis [10] [11] show that currently available inspection methods are mainly used for source code reviews. This is surprising and can be explained by the lack of tools that fully support software inspections [12] [13], especially in dealing with specific artifact types and locating potential defects in this artifacts. Furthermore, high inspection costs due to the resource-intensive nature of reviews and tedious searching, sorting or checking tasks often restrain the application of software inspections [11].

In this paper we present a tool-based approach that tries to identify potential document quality defects. This tool-based analysis relies on best practices for software documentation. In section II we give an overview of related work in the context of software inspection and document quality defect management tools. Section III shows how documentation best practices can be used to identify document quality defects. In section IV we present our developed tool-based document quality defect detection approach. Section V gives an overview of the results of an empirical study where we used

our approach to detect document quality defects in real-world project documentation. Finally, in section VI we give a conclusion and discuss further work.

II. RELATED WORK

In this section we give an overview of existing tools that can be used in the document inspection process. As there are different criteria for categorizing and distinguishing inspection tools [12] [13], we focus on tools that directly address data defects, i.e. tools that enable locating potential quality defects in the documents. Following, we also discuss work of software engineering domains apart from software inspections, but enable comprehensive document quality analysis and assessments. However, tools that support e.g. the collaborative inspection process or the process improvement are out of scope of this work and will not be discussed in this section.

Wilson, Rosenberg and Hyatt [14] present an approach for the quality evaluation of natural language software requirements specifications, introducing a quality model containing eleven quality attributes and nine quality indicators. Furthermore, a tool called ARM (Automatic Requirements Measurement) is described, which enables performing analysis of natural language requirements against the quality model with the help of quality metrics. Lami and Ferguson [15] describe a methodology for the analysis of natural language requirements based on a quality model that addresses the expressiveness, consistency and completeness of requirements. Moreover, to provide support for the methodology on the linguistic level of requirements specifications, they present the tool QuARS (Quality Analyzer of Requirements Specifications) [16]. Further tools that also support the automatic analysis of natural language requirements documents are described e.g. by Jain, Verma, Kass and Vasquez [17] and Raven [18]. However, all these tools are limited to the analysis of requirements specifications that are available as plain text documents. Although, the quality of a software project strongly depends on its requirements, there are a number of additional document types and formats that have to be considered throughout the software development life-cycle.

Farkas, Klein and Röbig [19] describe an automated review approach for ensuring standard compliance of multiple software artifacts (e.g. requirements specifications, UML models, SysML models) for embedded software using a guideline checker called Assessment Studio. The tool performs checks on XML-based software artifacts by using rules formalized in LINQ (Language Integrated Query) [20]. As they use XML as a common file-basis, their approach is not limited to one specific document type or format. Moreover, traceability checks of multiple software artifacts are facilitated. Nödler, Neukirchen and Grabowski [21] describe a comparable XQuery-based Analysis Framework (XAF) for assuring the quality of various software artifacts. XAF enables the specification of XQuery analysis rules, based on standardized queries and pattern matching expressions. In contrast to the approach presented in [19], XAF uses a facade layer for transforming XQuery rules to the individual XML representation of the underlying software artifact. As a result of this layered architecture, XAF enables the creation of re-usable analysis rules that are independent from the specific target software artifact.

III. DOCUMENTATION BEST PRACTICES FOR MEASURING DOCUMENT QUALITY

International documentation and requirements specification standards like NASA-STD-2100-91 [22], IEEE Std 830-1998 [23], IEEE Std 1063-2001 [24], ISO/IEC 18019:2004 [25], and ISO/IEC 26514:2008 [26] provide best practices and guidelines for information required in software documentation. Most of these documentation standards focus on guidelines for technical writers and editors producing manuals targeted towards end users. Hargis et al. [27] focus on quality characteristics and distinguish nine quality characteristics of technical information, namely “task orientation”, “accuracy”, “completeness”, “clarity”, “concreteness”, “style”, “organization”, “retrieveability”, and “visual effectiveness”. Moreover, they provide checklists and a procedure for reviewing and evaluating technical documentation according to these quality characteristics. In order to determine the quality of project documents, Arthur and Stevens [28] identified in their work four characteristics (“accuracy”, “completeness”, “usability”, and “expandability”) that are directly related to the quality of adequate documentation. Nevertheless, documentation quality is difficult to measure. Therefore, Arthur and Stevens [28] refined each documentation quality attribute to more tangible documentation factors, which can be measured by concrete quantifiers. In order words, similar to static code analysis, some quality aspects of project documentation can be determined by means of metrics. Moreover, we think that violations of documentation best practices or generally accepted documentation guidelines can also serve as measurable quantifiers. Consequently, violations of defined rules, which represent such best practices or guidelines, can be used for determining quality defects of documentation.

So far we have identified and specified more than 60 quantifiable documentation rules. Most of these document quality rules cover generally accepted best practices according to the documentation and requirements standards mentioned above. In order to get a better understanding about document quality rules, we show four typical examples. Furthermore we try to emphasize the importance of these rules for software projects, as well as the challenges of checking them automatically.

A. Adhere to document naming conventions

Each document name has to comply with naming conventions. The usage of document naming conventions helps recognizing the intended and expected content of a document from its name, e.g., requirements document, design specification, test plan. A consistent naming scheme for documents is especially important in large-scale software

projects. However, defining generally accepted naming conventions for arbitrary projects is not always simple and requires support for easy configuration of a specific project.

B. Each document must have an author

Each document must explicitly list its authors, as content of documents without explicit specified authors cannot be traced back to its creators. This is important e.g., for requirements documents in order to clarify ambiguous specifications with the authors. However, identifying document content particles describing author names is difficult and needs sophisticated heuristics.

C. Ensure that each figure is referenced in the text

If a figure is not referenced in the text, this reference might either be missing or the intended reference might be wrong. Typically, many figures are not self-explanatory and have to be described in the text. It is good style (e.g., in a software design document), to explain a UML sequence diagram or a class diagram. In order to make this explanation readable and consistent, it must always be clear which specific UML artifacts are explained in the text.

D. Avoid duplicates in documents

Within one document duplicated paragraphs exceeding a defined length should be avoided and explicitly be referenced instead, as duplicates make it difficult to maintain the document content. Therefore, word sequences of a specified length that are similar (e.g., defined by a percent value) to other word sequences in the same document violate this rule. However, the defined length of the word sequence strongly depends on the document type and differs from project to project.

IV. THE AUTOMATED DOCUMENT QUALITY DEFECT DETECTION APPROACH

As shown in section III, the identification of documentation defects in software development documents can rely on finding violations of document quality rules, which represent generally accepted documentation best practices and guidelines. However, manual checks of these rules can be very resource and time consuming, especial in large-scale software projects. Due to this, we developed a document quality defect detection tool, which checks software development documents against implemented document quality rules.

Similar to existing static code analysis suites for source code, our tool analyzes document information elements to find out, whether documents adhere to explicitly defined documentation best practices. In contrast to approaches and tools mentioned in section II, our document quality defect detection tool is not restricted to elementary lexical or linguistic document content analysis. Furthermore, it is also not limited to specific software development artifacts but covers the range from requirements across system, architecture and design, up to test specifications. The introduction of open, standardized document formats like Office Open XML [29] or

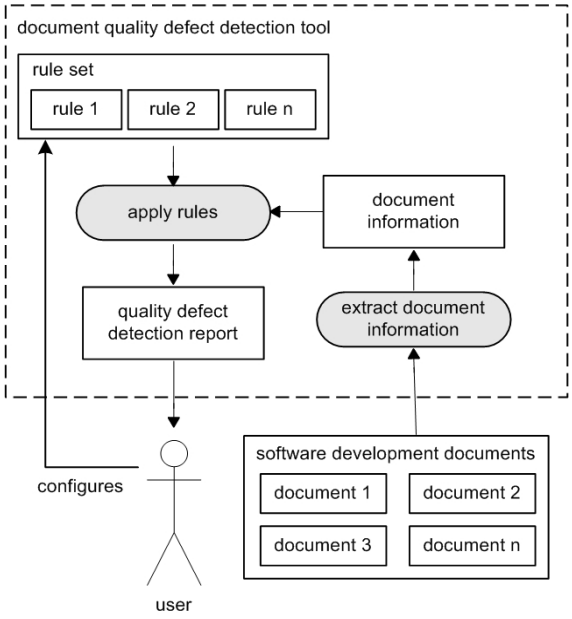


Figure 1. Conceptual overview of the document quality defect detection tool usage process

Open Document Format [30] has enabled the extraction of document information in a way, which is beyond unstructured text. In fact, our tool facilitates beside content quality analysis also the check of document metadata like directory information and version information. The use of standardized and structured document models also allows checking more specific content attributes based on the meaning of specific document particles. Moreover, it enables traceability checks to prove document information for project wide consistency. In order to support inspectors in their task, the tool can therefore be used to automatically check cross-references within the document under inspection as well as from the document under inspection to other related documents. The latter aspect is especially important for identifying missing or broken relations between e.g., design documents and software requirements specifications.

Fig. 1 gives a conceptual overview of our developed tool and describes the process of quality defect detection. First of all, the tool user (e.g. project manager, software inspector, quality manager) has to choose the software development documents as well as the document quality rules respectively rule sets, which will be used for the defect detection. If necessary, the selected rules can be configured by the user to meet defined and project specific documentation requirements. After the user has started the tool, all relevant information is extracted from the selected software development documents. The information is represented as a hierarchical data structure containing information of specific document elements (e.g. sections, paragraphs, references, figures, sentences). In a next step, each rule will be applied onto this document information to check whether the document adheres to the rule conditions. Finally, all detected potential document quality defects are linked to the original document to provide a comprehensive quality defect detection report to the user.

As software development documents can exist in many different document formats, considering each format for automated quality defect detection with our tool is challenging. Due to this we developed in a first step a tool that is applicable for Open Office XML documents [29] and Microsoft Office Binary Format documents [31], as these are one of the most commonly used formats for software development documents. Furthermore, these standardized document formats provide access to particular document information that is required, as some rules are applied on specific document elements. Consequently, a traversal strategy to visit all these elements is needed. Due to this, we have implemented the visitor pattern [32] [33] for our tool. Using this pattern, which provides a methodology to visit all nodes of a hierarchical data structure, enables applying rules on each specified element of the extracted document information. A similar rule-checking mechanism is used by the Java source code measurement tool PMD [34]. However, instead of using software development rules for source code, our document quality defect detection tool uses this methodology in order to check software development documents by means of easily adaptable and highly configurable document quality rules.

V. FEASIBILITY STUDY OF THE AUTOMATED DOCUMENT QUALITY DEFECT DETECTION APPROACH

This section describes the results of a feasibility study conducted to test, whether an automated quality defect detection tool using 24 document quality rules is able to reveal additional documentation defects human inspectors did not find before. Furthermore the trustworthiness of these quality rules is shown as well as the effort to fix documentation defects and rule settings. Before, we will give in (A) a brief description of the software project documentation we used in our study and in (B) an overview of the applied document quality rules.

A. Description of the used software project documentation

In order to get appropriate software development documents for our feasibility study, we used project documents of a real-world software project. The software as well as the associated documentation was developed by Siemens AG Corporate Technology in several iterations using a semi-formal development process. The software is used for monitoring the communication and control flow in distributed applications. As we have our focus on the quality of software development documents, more technical or organizational background information of the project is not necessary for the purpose of our study.

TABLE I. SOFTWARE PROJECT DOCUMENT FORMAT TYPES

document format type	no. documents in project
DOC	124
XLS	11
PPT	37
PDF	7

The entire documentation of the software project contains of 179 software development documents of four different document format types. As it is shown in Table I, more than two-third of them are Microsoft Office Word Binary Format documents. However, some of them are internal documents with intentionally lower quality. Due to this, we used a set of 50 officially published Microsoft Word project documents consisting of different document types (requirements specifications, systems specifications, concept analysis, market analysis, delta specifications, function lists, user documentation, etc.) as objects of analysis for our feasibility study. These 50 documents should meet high documentation quality standards and are already checked by software inspectors. Therefore, they testify to be of a high maturity level and ready to be checked by our tool.

B. Applied document quality rules

Following, we list and give a short description for all document quality rules we used in our study and motivate their importance for software development documents. However, the used rules settings are not discussed in this work.

- *ADNC - Adhere to Document Naming Conventions*: Each software development document name has to comply with explicitly specified naming conventions, as project members can better grasp the document content if documents follow a defined project-wide document naming scheme.
- *ADNS - Avoid Deeply Nested Sections:* Documents should not contain a deeply nested section hierarchy. Particularly in software development documents the content structure should be flat, simple and clear in order to support clarity.
- *ADUP - Avoid Duplicates in Document:* Similar to duplicated source code, within software development documents duplicated paragraphs exceeding a defined length (number of characters) should be omitted and are better referenced explicitly. Otherwise the document content will be more difficult to maintain.
- *AES - Avoid Empty Sections:* Each section of a software development document must contain at least one sentence, otherwise the content may not be complete or lacks of clarity and conciseness.
- *AESD - Avoid Extremely Small Documents:* Extremely small software development documents are indicators for unfinished content or for a bad project-wide document structure, as small software development documents might be better combined to larger document of reasonable size.
- *AIDOC - Avoid Incomplete Documents:* Particularly in later phases of the software development process documents should contain all information that is required. Therefore, documents that are formally incomplete, i.e., contain phrases like “TBD” or “TODO” are not yet complete by definition.

- *ALS - Avoid Long Sentences:* Identify those sentences in a project document that exceed a given length, where length is expressed by the number of words contained in the sentence. Long sentences harm the readability of e.g. requirements specifications or test plans and are therefore indicators for difficult to understand content of software development documents.
- *AULD - Avoid Ultra Large Documents:* Ultra-large software development documents should be avoided as they are more difficult to maintain and to keep consistent. Furthermore, it is harder to check whether all information needed is present.
- *ARHT - Avoid Repeated Heading Text:* In a software development document paragraphs of a section should not only consist of a copy of the heading text, as this a indicator of a underspecified and incomplete section
- *ASPE - Avoid Spelling Errors:* Each software development document should be free of spelling errors, regardless whether it is written in one language or contains a mix of languages.
- *ATSS - Adhere To Storage Structure:* Each software development document should be put in the right place of the storage system, i.e. it should typically be stored in a directory according to project-wide rules (typically for different types and/or phases of the software development process).
- *DESOR - Define Expected Skills Of Readers:* For each software development document the skills of readers should be explicitly defined, as depending on the skills of readers, the content of the software development document has to be presented in a different way. So, depending on the expected skills of the readers data might be presented more formally using e.g., UML, or must definitely avoid any formalisms.
- *DMHA - Document Must Have Author:* Each software development document must explicitly list its authors, as in the case of changes each document has to be traceable to its creators. Therefore, this rule is violated, if there is no author defined in the document meta-information and no key word is found that indicates the existence of an author name.
- *DMHV - Document Must Have Version Id:* Similar to source code each document in a software project should have an explicit version identifier.
- *DMHVH - Document Must Have Version History:* In order to keep software development documents comprehensible, each document must provide a version history that roughly outlines the changes over time (versions) during the entire software development process.
- *DMS* - Document *Must have a State:* Each software development document should outline its defined state (e.g., draft, in review, final, customer approved), in order to present the current document state to the project members.
- *ECNF - Ensure Continuous Numbering of Figures:* In software development documents ascending numbering of figures improves the quality of documentation, as this contributes to a higher consistency and comprehensibility of the documents.
- *ECNT - Ensure Continuous Numbering of Tables:* In software development documents an ascending numbering of tables improves the document quality, as this leads to higher consistency and comprehensibility of the document.
- *EFRT - Ensure that each Figure is Referenced in the Text:* Each figure has to be referenced in the text of software development documents; otherwise it is incoherent or might be ambiguous.
- *ETRT - Ensure that each Table is Referenced in the Text:* Each table has to be referenced in the text of software development documents; otherwise it is incoherent or might be ambiguous.
- *FMHC - Figures Must Have a Caption:* Each figure in a software development document must have a caption in order to express the visualized topics linguistically; otherwise it may be ambiguous for the readers.
- *PIFF - Provide Index For Figures:* If a software development document contains figures, there must be an index listing all figures in order to keep information quickly retrievable for all project members.
- *PIFT - Provide Index For Tables:* If a software development document contains tables, there must be an index listing all tables in order to keep the information quickly retrievable for all project members.
- *TMHC – Tables Must Have a Caption:* Each table in a software development document must have a caption in order to express the visualized data of the table linguistically; otherwise it may be ambiguous for the readers.

C. Violations

In this section we give an overview of the results of our software development document defect detection analysis.

TABLE II. DEFECT DETECTION TOOL RESULTS

no. documents analyzed	50
no. document quality rules	24
total no. violations found	8,955
avg. false positive rate per rule	0.172

In our feasibility study 50 project documents were automatically checked by 24 document quality rules, which revealed a total number of 8,955 violations. For these findings we determined an average false positive rate per rule of 17.2 percent and a false negative rate per rule of 0.4 percent. False positive findings are (in our case) over-detected defects that are no documentation defects in the sense of human software inspectors. On the other hand, false negative findings are defects that have not been found by our tool but that are definitely documentation defects in the sense of human software inspectors.

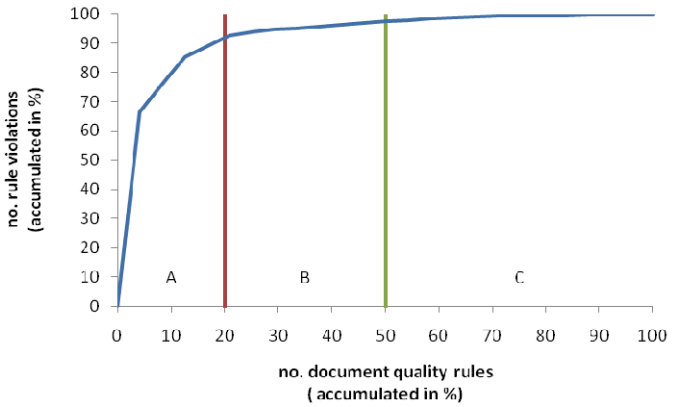


Figure 2. Rule violations per document quality rule distribution

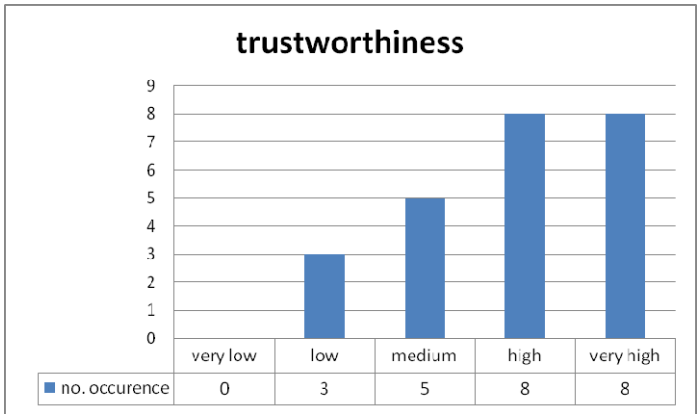


Figure 3. Trustworthiness of all applied document quality rules

TABLE III. RESULTS OVERVIEW PER RULE

rule	no. violations	false positive rate	false negative rate
ADNC	11	0	0
ADNS	14	0	0
ADUP	823	0	0
AES	337	0.033	0
AESD	30	0.933	0
AIDOC	0	0	0.020
ALS	49	0	0
AULD	9	0	0.100
ARHT	13	0.846	0
ASPE	5,956	0.602	0
ATSS	25	0	0
DESOR	50	0	0
DMHA	0	0	0.040
DMHV	21	0	0
DMHVH	14	0	0
DMS	48	0	0.040
ECNF	43	0.488	0
ECNT	46	0.326	0
EFRT	106	0.274	0
ETRT	75	0.160	0
FMHC	329	0.365	0
PIFF	50	0	0
PIFT	50	0	0
TMHC	856	0.093	0

During our investigations we also found out that the violations per rule are unequally distributed. As shown in Table III, the rules ADUP, AES, ASPE, FMHC and TMHC identified more than 300 violations each. Due to this, we accumulated the number of violations found by these five rules and compared it with the total number of violations. Consequently, as it can be seen in the ABC analysis diagram in Fig. 2, we revealed that these five document quality rules are responsible for more than 90 percent of all thrown violations.

D. *Trustworthiness*

The trustworthiness of a rule specifies how reliable the detection of a violation is. We classify trustworthiness into:

- *very low:* There is too much over- and/or under-detection in order to rely on the results.
- *low:* There is significant over- and under-detection.
- *medium:* Most issues are found, but there is over-detection.
- *high:* Almost no over- and under-detection. Very reliable findings.
- *very high:* No known over- or under-detection. Absolutely reliable findings.

As a result of this classification scheme, a main factor to determine the trustworthiness for a document quality rule is its false positive rate. Indeed, we also take false negative findings, as far as possible to indentify, and known weaknesses of the rule implementation into account, i.e., a rule with a false positive rate of 0.0 and/or a false negative rate of 0.0 does not implicitly have to have a trustworthiness rating of ‘very high’.

As shown in Fig. 3, we rated the trustworthiness of the document violations for eight of our 24 applied rules with ‘very high’, i.e., these violations are very reliable.

TABLE IV. ‘VERY HIGH’ TRUSTWORTHY RULES

ADNC	Adhere to Document Naming Conventions
ADNS	Avoid Deeply Nested Sections
ADUP	Avoid Duplicates in Document
ALS	Avoid Long Sentences
ATSS	Adhere To Storage Structure
DMHVH	Document Must Have Version History
PIFF	Provide Index For Figures
PIFT	Provide Index For Tables

Furthermore, we also determined for eight document quality rules a ‘high’ trustworthiness, as we identified almost no over- or under-detection for this rules. As a result of this

more than two-third of our rules are identified to be 'very high' or 'high' trustworthy.

TABLE V. 'HIGH' TRUSTWORTHY RULES

AES	Avoid Empty Sections
AIDOC	Avoid Incomplete Documents
AULD	Avoid Ultra Large Documents
DESOR	Define Expected Skills Of Readers
DMHA	Document Must Have Author
DMHV	Document Must Have Version Id
ETRT	Ensure that each Table is Referenced in the Text
TMHC	Table Must Have a Caption

However, our feasibility study also revealed three rules with a 'low' trustworthiness.

TABLE VI. 'LOW' TRUSTWORTHY RULES

AESD	Avoid Extremely Small Documents
ARHT	Avoid Repeated Heading Text
ASPE	Avoid Spelling Errors

These rules have to deal with a false positive rate of more than 60 percent, e.g. most of the ASPE violations are thrown as domain specific terms or abbreviations are falsely identified as misspelled words. Nevertheless, some of the violations of these three rules are informative as we think that, although there is much over- and under-detection, they can be categorized as 'low' trustworthy. Moreover, we think that small rule improvements, e.g. adding the usage of a domain specific dictionary for the ASPE rule, would lead to a higher trustworthiness.

E. *Effort to fix defects*

The effort to fix true positive findings specifies how much is needed to spent for removing a defect (qualitatively):

- *low:* Only some local lines in a document have to be changed.
- *medium:* Document-wide changes are necessary.
- *high:* Project-wide document changes are necessary.

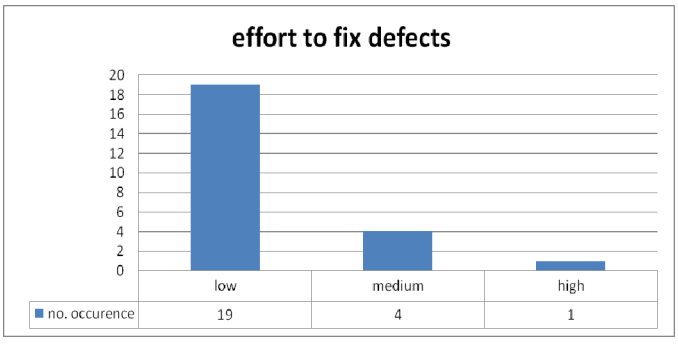


Figure 4. 'Effort to change defects' of all applied document quality rules

As shown in Fig. 4, most violations thrown by 19 of our 24 applied rules affect only some lines in the documents, i.e. these defects can be quickly corrected and represent easy wins. Moreover, for fixing the defects of four of our rules we determined that document-wide changes are required.

TABLE VII. 'MEDIUM' EFFORT TO FIX DEFECTS

ADNS	Avoid Deeply Nested Sections
ADUP	Avoid Duplicates in Document
AESD	Avoid Extremely Small Documents
DMHVH	Document Must Have Version History

Nevertheless, during our feasibility study we also determined that all true positive AULD violations lead to project-wide document changes. In this case, high effort is needed as an ultra large document has to be split into separate documents. Furthermore, all references are affected and have to be checked for correctness. It is very hard to determine whether defects of a specific rule generally affect only some lines in a document or the entire software project, as e.g. small changes in some lines can also lead to broken references in other documents.

F. *Effort to change settings*

The effort to adapt configuration settings of the rules to the needs of the specific project specifies how much effort is needed to spent for adapting the rule configurations, before the document defect detection tool can be applied:

- *low:* Nothing or very small adaptations are necessary in a settings file.
- *medium:* Some lines have to be changed in a settings file. Some knowledge of the analyzed project documents is necessary to define, e.g., suitable regular expressions.
- *high:* Settings files have to be changed considerably. Detailed information of the project document content and structure is necessary to define, e.g., suitable regular expressions.

As stated in Fig. 5, more than two-thirds of all applied document quality rules do not need considerable effort to be

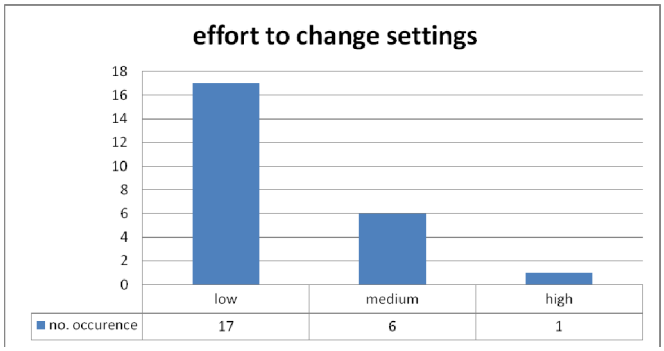


Figure 5. 'Effort to change settings' of all applied document quality rule

suitably configured. In order to correctly configure six of our rules it is necessary to have some further knowledge in specifying correct regular expressions.

TABLE VIII. 'MEDIUM' EFFORT TO CHANGE SETTINGS

ADNC	Adhere to Document Naming Conventions
ASPE	Avoid Spelling Errors
ATSS	Adhere To Storage Structure
DMHV	Document Must Have Version Id
EFRT	Ensure that each Figure is Referenced in the Text
ETRT	Ensure that each Table is Referenced in the Text

Furthermore, it is required to have an overview of the document structure and document content. However, to correctly configure the DESOR rule (effort to change settings = 'high'), there must be also some knowledge of the used expressions and languages in order to identify and extract the specific document content properties that define the skills of readers.

VI. CONCLUSION AND FURTHER WORK

Empirical studies show that tool support can significantly increase the performance of the overall software inspection process [10][11][12][13]. However, most available software inspection tools are optimized for code inspections, which usually provide support for plain text documents, only. Due to this they are inflexible with respect to different artifact types and limit inspectors in their work. For natural language text, inspection tools cannot fully replace human inspectors in detecting defects. Nevertheless, software inspection tools can be used to make defect detection tasks easier [11]. Encouraged by this, we developed a tool-based quality defect detection approach to support the inspection process by checking documentation best practices in software development documents.

International documentation and specification standards [22] [23] [24] [25] [26] define a set of generally accepted documentation best practices. Furthermore, checklists and reviewing procedures [27] are widely used as well as documentation quantifiers in order to check specific documentation characteristics representing quality aspects [28]. As a result of these studies we came to the conclusion, that measurable document quality rules expressing best practices can also be used to detect defects in software development documents and to help enhancing documentation quality. So far we have implemented a document quality defect detection tool, which is applicable on Office Open XML documents [29] and Microsoft Office Binary Format documents [31]. The tool allows checking, whether software development documents adhere to explicitly defined document quality rules. In a feasibility study we showed that our automatic defect detection tool is capable of finding additional uncovered significant documentation defects that had been overlooked by human inspectors.

During our analysis, we automatically checked 50 Microsoft Office Word documents of a real-world software project with 24 document quality rules. The tool revealed 8,955 violations with an average false positive rate per rule of 17.2 percent and an average false negative rate per rule of 0.4 percent. As our study shows, two-thirds of all applied document quality rules were rated with a 'high' or 'very high' trustworthiness. Furthermore, it has been pointed out that most of the violations found can be easily removed (effort to change defect = 'low'), as they often only affect some few lines.

In our feasibility study we determined that nearly 75 percent of all rules did not need any further configuration changes before they could be suitably applied to software development documents. Nevertheless, seven rules had to be adapted to project specific document conventions before they could be applied. In the case of the project documentation used for our study the configuration of these rules took us approximately six hours, as we were not familiar with the conventions defined for the document naming and content structure. However, we saw that after the rules had been suitably configured, the trustworthiness of the rule violations rose considerably, i.e., the configuration effort well paid-off.

In a next step, we will apply our document quality defect detection tool on the documents of additional software projects to improve the implementation of the rules with an emphasis on reducing the false positive rate and to validate the results of our feasibility study in more detail. Moreover, as we have seen that some of our rules are applicable for most technical documents, we also want to implement some document quality rules that are even more specific for software development rules. For instance, we will add rules that deal with domain specific terms and glossaries used in software documents or the traceability of references between various software development documents (of different software life-cycle phases).

We currently also work on transferring Adobe PDF documents in a way that the already developed document quality rules for the Office Open XML documents and Microsoft Office Binary Format documents can be used without changes. As a result of this, we think that the definition of an abstract document structure that separates the rules from the underlying software artifacts is essential. Consequently, this would enable a much easier development of rules that can be applied on elements of a general document model, as there is no need to deal with the complexity of specific document formats for the rule development. Furthermore, we recognized that our rules are too loosely grouped. From our experience with rules in the context of code quality [35], we will develop a quality model that allows a systematic clustering of rules by means of quality attributes. This will give us the possibility to evaluate document quality on more abstract levels, like readability or understandability of a document.

ACKNOWLEDGMENTS

We would like to thank Siemens AG Corporate Technology for supporting our empirical investigations by providing us with software development documentation data in order to conduct our feasibility study and test our approach.

REFERENCES

[1] B. W. Boehm, Software Engineering. Barry W. Boehm's lifetime contributions to software development, management, and research. Hoboken, N.J., Wiley-Intersience, 2007.

[2] L. Briand, K. E. Emam, O. Laitenberger, and T. Fussbroich, Using Simulation to Build Inspection Efficiency Benchmarks for Development Projects. International Conference on Software Engineering, IEEE Computer Society, 1998.

[3] J. C. Chen and S. J. Huang, "An empirical analysis of the impact of software development problem factors on software maintainability," in Journal of Systems and Software. Elsevier Science Inc., 2009, vol. 82, pp. 981-992.

[4] M. E. Fagan, "Design and code inspections to reduce errors in program development," in IBM Systems Journal. vol. 15 (3), 1976, pp. 182-211.

[5] T. Gilb and D. Graham, Software Inspection. Addison-Wesley Publishing Company, 1993.

[6] T. Thelin, P. Runeson, and C. Wohlin, "An Experimental Comparison of Usage-Based and Checklist-Based Reading," in IEEE Trans. Software Engineering, vol. 29, no. 8, Aug. 2003, pp. 687-704.

[7] T. Thelin, P. Runeson, C. Wohlin, T. Olsson, and C. Andersson, "Evaluation of Usage-Based Reading-Conclusions after Three Experiments," in Empirical Software Engineering: An Int'l J., vol. 9, no. 1, 2004, pp. 77-110.

[8] F. Shull, I. Rus, and V. Basili, "How Perspective-Based Reading Can Improve Requirements Inspections," in Computer, vol. 33, no. 7, July 2000, pp. 73-79.

[9] J. Carver, F. Shull, and V.R. Basili, "Can Ovservational Techniques Help Novices Overcome the Software Inspection Learning Curve? An Empirical Investigation," in Empirical Software Engineering: An Int'l J., vol. 11, no. 4., 2006, pp. 523-539.

[10] O. Laitenberger and J.-M. DeBaud, "An encompassing life cycle centric survey of software inspection," in Journal of Systems and Software. vol. 50, 2000, pp. 5-31.

[11] S. Biff; P. Grünbacher, and M. Halling, "A family of experiments to investigate the effects of groupware for software inspection," in Automated Software Engineering, Kluwer Academic Publishers, vol. 13, 2006, pp. 373-394.

[12] H. Hedberg and J. Lappalainen, A Preliminary Evaluation of Software Inspection Tools, with the DESMET Method. Fifth International Conference on Quality Software, IEEE Computer Society, 2005, pp. 45-54.

[13] V. Tenhunen and J. Sajaniemi, An Evaluation of Inspection Automation Tools. International Conference on Software Quality, Spinger-Verlag, 2002, pp. 351-362.

[14] W. M. Wilson, L. H. Rosenberg, and L.E. Hyatt, Automated quality analysis of Natural Language Requirement specifications. PNSQC Conference, October 1996.

[15] G. Lami and R. W. Ferguson, "An Empirical Study on the Impact of Automation on the Requirements Analysis Process," in Journal of Computer Science Technology. vol. 22, 2007, pp. 338-347.

[16] G. Lami, QuARS: A tool for analyzing requirements. Software Engineering Institute, 2005.

[17] P. Jain, K. Verma, A. Kass, and R. G. Vasquez, Automated review of natural language requirements documents: generating useful warnings with user-extensible glossaries driving a simple state machine. Proceedings of the 2nd India software engineering conference, ACM, 2009, pp. 37-46.

[18] Raven: Requirments Authoring and Validation Environment, www.ravenflow.com.

[19] T. Farkas, T. Klein, H. Röbig, "Application of Quality Standards to Mutliple Artifacts with a Universal Compliance Solution", in Model-Based Engineering of Embedded Real-Time Systems. International Dagstuhl Workshop, Dagstuhl Castle, Germany, 2007.

[20] Microsoft Developer Network: The LINQ Project, http://msdn.microsoft.com/en-us/netframework/aa904594.aspx

[21] J. Nödler, H. Neukirchen, and J. Grabowski, "A Flexible Framework for Quality Assurance of Software Artefacts with Applications to Java, UML, and TTCN-3 Test Specifications" in Proceedings of the 2009 International Conference on Software Testing Verification and Validation. IEEE Computer Society, 2009, pp. 101-110.

[22] NASA Software Documentation Standard, NASA-STD-2100-91. National Aeronautics and Space Administration, NASA Headquarters, Software Engineering Program, July, 1991.

[23] IEEE Recommended Practice for Software Requirements Specifications, IEEE Std 830-1998. 1998.

[24] IEEE Standard for Software User Documentation, IEEE Std 1063-2001. 2001

[25] ISO/IEC 18019:2004: Software and system engineering - Guidelines for the design and preparation of user documentation for application software, 2004.

[26] ISO/IEC 26514:2008: Systems and software engineering - Requirements for designers and developers of user documentation, 2008.

[27] G. Hargis, M. Carey, A. K. Hernandez, P. Hughes, D. Longo, S. Rouiller, E. Wilde, Developing Quality Technical Information: A Handbook for Writers and Editors, 2nd ed. IBM Press, 2004.

[28] J. D. Arthur, and K. T. Stevens, Document Quality Indicators: A Framework for Assessing Documentation Adequacy. Virginia Polytechnic Institute, State University, 1990.

[29] ISO/IEC 29500:2008. Information technology – Document description and processing languages – Office Open XML File Formats Open Document Format, 2008.

[30] ISO/IEC 26300:2006. Information technology- Open Document Format for Office Applications (OpenDocument), 2006.

[31] Microsoft Office Binary File Format: http://www.microsoft.com/interop/docs/OfficeBinaryFormats.mspx

[32] P. Buchlovsky and H. Thielecke, "A Type-theoretic Reconstruction of the Visitor Pattern. Electronic Notes," in Theoretical Computer Science. vol. 155, 2006, pp. 309 - 329.

[33] B. C. Oliveira, M. Wang, and J. Gibbons, The visitor pattern as a reusable, generic, type-safe component. Proceedings of the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications. ACM, 2008, pp. 439-456.

[34] T. Copeland, PMD Applied. Centennial Books, 2005.

[35] R. Plösch, H. Gruber, A. Hentschel, Ch. Körner, G. Pomberger, S. Schiffer, M. Saft, and S. Storck, "The EMISQ Method and its Tool Support - Expert Based Evaluation of Internal Software Quality," in Journal of Innovations in Systems and Software Engineering. Springer London, vol. 4(1), March 2008.

Design Pattern Detection using Software Metrics and Machine Learning

Satoru Uchiyama
Hironori Washizaki
Yoshiaki Fukazawa
Dept. Computer Science and Engineering
Waseda University
Tokyo, Japan
s.uchiyama1104@toki.waseda.jp
washizaki@waseda.jp
fukazawa@waseda.jp

Atsuto Kubo
Aoyama Media Laboratory
Tokyo, Japan
kubo@nii.ac.jp

***Abstract*—The understandability, maintainability, and reusability of object-oriented programs could be improved by automatically detecting well-known design patterns in programs. Many existing detection techniques are based on static analysis and use strict conditions composed of class structure data. Hence, it is difficult for them to detect design patterns in which the class structures are similar. Moreover, it is difficult for them to deal with diversity in design pattern applications. We propose a design pattern detection technique using metrics and machine learning. Our technique judges candidates for the roles that compose the design patterns by using machine learning and measurements of metrics, and it detects design patterns by analyzing the relations between candidates. It suppresses false negatives and distinguishes patterns in which the class structures are similar. We conducted experiments that showed that our technique was more accurate than two previous techniques.**

Keywords—component; Object-oriented software, Design pattern, Software metrics, Machine learning

I. INTRODUCTION

Design patterns (hereafter, patterns) are defined as descriptions of communicating classes that form a common solution to a common design problem. Gang of Four (GoF) patterns [1] are representative patterns for object-oriented software. Patterns are composed of classes that describe the roles and abilities of objects. For example, Figure 1 shows one GoF pattern named the `State pattern`. This pattern is composed of roles named `Context`, `State`, and `ConcreteState`. The use of patterns enables software development with high maintainability, high reusability, and improved understandability, and it facilitates smooth communications between developers.

Programs implemented by a third party and open source software may take a lot of time to understand, and patterns may be applied without explicit class names, comments, or attached documents in existing programs. Thus, pattern detection improves the understandability of programs. However, manually detecting patterns in existing programs is inefficient, and patterns may be overlooked.

Many studies on using automatic pattern detection to solve the above problems have used static analysis. However, static analysis has difficulty identifying patterns in which class structures are similar and patterns with few features. In addition, there is still a possibility that software developers might overlook patterns if they use strict conditions like the class structure analysis, and if the applied patterns vary from the intended conditions even a little.

We propose a pattern detection technique that uses software metrics (hereafter, metrics) and machine learning. Although our technique can be classified as a type of static analysis, unlike previous detection techniques it detects patterns by using identifying elements derived by machine learning based on measurement of metrics without using strict condition descriptions (class structural data, etc.). A metric is a quantitative measure of a software property that can be used to evaluate software development. For example, one such metric, number of methods (NOM), refers to the number of methods in a class [2]. Moreover, by using machine learning, we can in some cases obtain previously unknown identifying elements from combinations of metrics. To cover a diverse range of pattern applications, our method uses a variety of learning data because the results of our technique may depend on the kind and number of learning data used during the machine learning process. Finally, we conducted experiments comparing our technique with two previous techniques and found that our approach was the most accurate of the three.

II. PREVIOUS DESIGN PATTERN DETECTION TECHNIQUES AND THEIR PROBLEMS

Most of the existing detection techniques use static analysis [3][4]. These techniques chiefly analyze information such as class structures that satisfy certain conditions. If they vary from the intended strict conditions even a little, or two or more roles are assigned in a class, there is a possibility that developers might overlook patterns.

There is a technique that detects patterns based on the degrees of similarity between graphs of pattern structure and graphs of programs to be detected [3]. However,

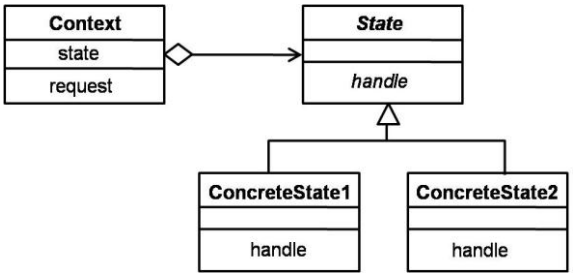


Figure 1. State pattern

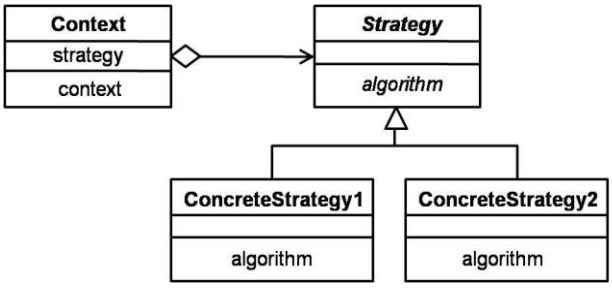


Figure 2. Strategy pattern

distinguishing the `State` pattern from the `Strategy` pattern is difficult because their class structures are similar (see Figure 1 and 2). Unlike this method, we distinguish the patterns to which the structure is similar by the identification of the roles from the quantity and the ratio of metrics by the machine learning. In addition, this technique [3] is available to the public as a web-based tool.

There is a technique that outputs pattern candidates based on features derived from metric measurements [5]. However, it requires manual confirmation; this technique can roughly identify pattern candidates, but the final choice depends on the developer's skill. Our technique detects patterns without manual filtering by metrics and machine learning but also by analyzing class structure information. Moreover, this technique uses general metrics concerning an object-oriented system without using metrics for each role. Our technique uses metrics that specialize in each role.

Another existing technique improves precision by filtering detection results using machine learning. This technique inputs measurements of the classes and methods of each pattern [6]. However, it uses the existing static analytical approach, whereas our technique instead uses machine learning throughout the entire process.

One current technique analyzes programs both before and after patterns are applied [7]. This method requires a revision history of the programs used. Our technique detects patterns by analyzing only the current programs.

Yet another approach detects patterns from the class structure and behavior of a system after classifying its patterns [8][9]. It is difficult to use, however, when patterns are applied more than once and when pattern application is diverse. In contrast, our technique copes well with both of these challenges.

Other detection techniques use dynamic analysis. These methods identify patterns by referring to the execution route information of programs [10][11]. However, it is difficult to analyze the entire execution route and use fragmentary class sets in an analysis. Additionally, the results of dynamic analysis depend on the representativeness of the execution sequences.

Some detection techniques use a multilayered (multiphase) approach [12][13]. Lucia et al. use a two-phase, static analysis approach [12]. This method has difficulty, however, in detecting creational and behavioral patterns because it analyzes pattern structures and source code level conditions. Guéhéneuc et al. use "DeMIMA," an approach that consists of three layers: two layers to recover an abstract model of the source code, including binary class relationships, and a third layer to identify patterns in the abstract model. However, distinguishing the `State` pattern from the `Strategy` pattern is difficult because their structures are identical. Our technique can detect patterns in all categories and distinguish the `State` pattern from the `Strategy` pattern using metrics and machine learning.

Finally, one existing technique detects patterns using formal OWL (Web Ontology Language) definitions [14]. However, false negatives arise because this technique does not accommodate the diversity in pattern applications. The technique [14] is available to the public via the web as an Eclipse plug-in.

We suppress false negatives by using metrics and machine learning to accommodate diverse pattern applications and to distinguish patterns in which the class structures are similar. It should be noted that only techniques [3], [14] discussed above have been released as publicly accessible tools.

III. OUR TECHNIQUE

Our technique is composed of a learning phase and a detection phase. The learning phase is composed of three processes, and the detection phase is composed of two processes, as shown in Figure 3. Each process is described below, with pattern specialists and developers included as the parties concerned. Pattern specialists mean persons that have the knowledge about the patterns. Developers mean persons that maintain the object-oriented software. Our technique currently uses Java as the program language.

[Learning Phase]

P1. Define Patterns

Pattern specialists determine the detectable patterns and define the structures and roles composing these patterns.

P2. Decide Metrics

Pattern specialists determine useful metrics to judge the roles defined in P1 by using the Goal Question Metric decision technique.

P3. Machine Learning

Pattern specialists input programs applied patterns into the metrics measurement system, and obtain measurements for each role. And specialists input these measurements into the machine learning simulator to learn. After machine learning they verify the judgment for each role, and if

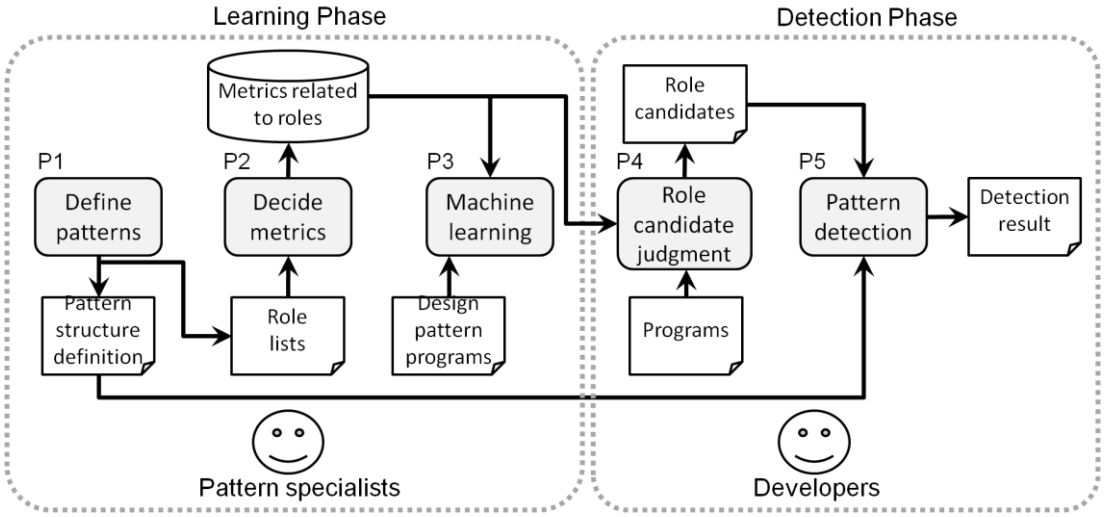


Figure 3. Process of our technique

the verification results are not good, they return to P2 and revise the metrics.

[Detection Phase]

P4. Role Candidate Judgment

Developers input programs to be detected into the metrics measurement system, and obtain measurements for each class. And developers input these measurements into the machine learning simulator. Machine learning simulator identifies role candidates.

P5. Pattern Detection

Developers input role candidates judged in P4 to the pattern detection system by using the pattern structure definitions defined in P1. This system detects patterns automatically. The structure definitions correspond to the letters P, R, and E of section III-B.

A. Learning Phase

P1. Define Patterns

Currently, our technique considers five GoF patterns (`Singleton`, `TemplateMethod`, `Adapter`, `State`, and `Strategy`) and 12 roles. The GoF patterns are grouped into creational patterns, structural patterns, and behavioral patterns. Our technique uses these patterns to cover all these groups.

P2. Decide Metrics

Pattern specialists decide on useful metrics to judge roles by using the Goal Question Metric decision technique [14] (hereafter, GQM). GQM is a top-down approach used to clarify relations between goals and metrics.

We experimented with judging roles by using general metrics without GQM. However, the machine learning results were unsatisfactory because the measurements of some metrics were irregular. Consequently, we chose GQM so that the machine learning could function appropriately by stable metrics in each role. With our technique, the pattern specialists set as a goal the accurate judgment of each role. To achieve this goal they defined a set of questions to be evaluated. Next, they decided on useful metrics to help answer the questions they had established. The pattern specialists decide metrics to identify roles by the quantity and the ratio of measurements. Therefore, they decide questions by paying attention to the attributes and operations of the roles by reading the description of the pattern definition. They decide simple metrics concerning the static aspect like structure first to improve the recall. However, the lack of questions might occur because GQM is qualitative. Therefore, if the machine learning results were unsatisfactory by irregular measurements of metrics, the procedure loops back to P2 to reconsider metrics also concerning behavior. Moreover, it will be possible to apply GQM to roles with new patterns in the future.

For example, Figure 4 illustrates the goal of making a judgment about the `AbstractClass` role in the `TemplateMethod` pattern. `AbstractClass` roles have abstract methods or methods using written logic that use abstract methods as shown in Figure 5. The `AbstractClass` role can be distinguished by the ratio of

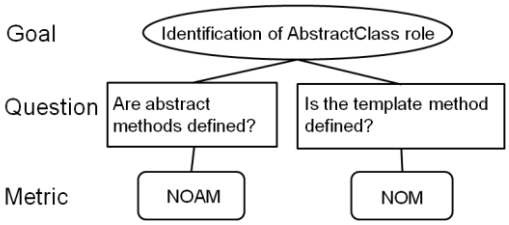


Figure 4. Example of GQM（AbstractClass role）

```
public abstract class AbstractDisplay {
    public abstract void open();
    public abstract void print();
    public abstract void close();
    public final void display() {
        open();
        for (int i = 0; i < 5; i++) {
            print();
        }
        close();
    }
}
```

Figure 5. Example of source code (AbstractClass role)

TABLE I. RESULTS OF APRLING GQM

Pattern	Role	Goal	Question	Metric
Singleton	Singleton	Identification of Singleton role	Is the static field defined?	NSF
			Is the constructor called from other class?	NOPC
			Is the method that return singleton instance?	NSM
Template Method	AbstractClass	Identification of AbstractClass role	Are abstract methods defined?	NOAM
			Is the template method defined?	NOM
	ConcreteClass	Identification of ConcreteClass role	Is the override method defined?	NORM
Adaper	Target	Identification of Target role	Are abstract methods defined?	NOAM
			Is the class defined as an interface?	NOI
	Adapter	Identification of Adapter role	Are override methods defined?	NORM
			Is Adaptee field defined?	NOF
				NOOF
	Adaptee	Identification of Adaptee role	Are methods used by Adapter role defined?	NOM
			Is the class referred by other classes?	NCOF
State	Context	Identification of Context role	Are methods to set states defined?	NOM
			Is State field defined?	NOF
				NOOF
	State	Identification of State role	Are abstract methods defined?	NOAM
			Is the class defined as an interface?	NOI
			Is the class referred by other classes?	NCOF
	Concrete State	Identification of ConcreteState role	Is the override method defined?	NORM
			Is the method that describes change state defined?	NOM
				NMGI
Strategy	Context	Identification of Context role	Are methods to set states defined?	NOM
			Is Strategy field defined?	NOF
				NOOF
	Strategy	Identification of Strategy role	Are abstract methods defined?	NOAM
			Is the class defined as an interface?	NOI
			Is the class referred by other classes?	NCOF
	Concrete Strategy	Identification of ConcreteStrategy role	Is the override method defined?	NORM

the number of methods to the number of abstract methods because with this role the former exceeds the latter. Therefore, the number of abstract methods (NOAM) and number of methods (NOM) are useful metrics for judging this role. Table I shows the results of applying GQM to all roles. The details of metrics are described in Table II of paragraph IV-*A*.

The previous technique [5] uses GQM, too. In this technique, the goal is set as "Recover design patterns". And this technique uses general metrics concerning an object-oriented system without deciding metrics at each role. On the other hand, our technique uses metrics that specialize in each role.

P3. Machine Learning

Machine learning is a technique that analyzes sample data by computer and acquires useful rules with which to make forecasts about unknown data. We used the machine learning so as to be able to evaluate patterns with a variety of application forms. Machine learning suppresses false negatives and achieves extensive detection.

Our technique uses a neural network [16] algorithm. A support vector machine [16] could also be used to distinguish a pattern of two groups by using linear input elements. However, we chose a neural network because it outputs the values to all roles, taking into consideration the dependency among the different metrics. Therefore, it can deal with cases in which one class plays two or more roles.

A neural network is composed of an input layer, hidden layers, and an output layer, as shown in Figure 6, and each layer is composed of elements called units. Values are given a weight when they move from unit to unit, and a judgment rule is acquired by changing the weights. A typical algorithm for adjusting weights is back propagation. Back propagation calculates an error margin between output result y and the correct answer T, and it sequentially adjusts weights from the layer nearest the output to the input layer, as shown in Figure 7. These weights are adjusted until the output error margin of the network reaches a certain value.

Our technique uses a hierarchical neural network simulator [17]. This simulator uses back propagation. The hierarchy number in the neural network is set to three, the number of units in the input layer and the hidden layer are set to the number of decided metrics, and the number of units of the output layer is set to the number of roles being judged.

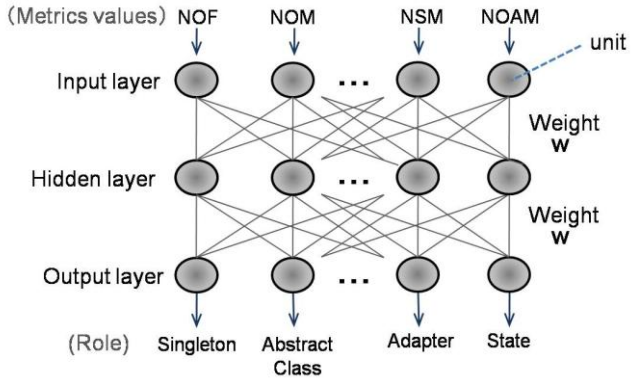


Figure 6. Neural network

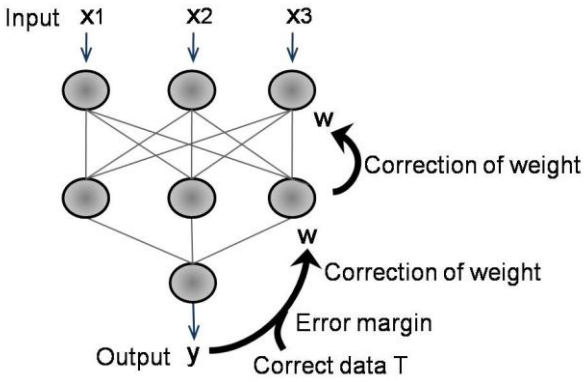


Figure 7. Back propagation

The input consists of the metric measurements of each role in a program to which patterns have already been applied, and the output is an expected role. Pattern specialists obtain measurements for each role by using metrics measurement system. And, specialists input these measurements into the machine learning simulator to learn. The learning repetitions cease when the error margin curve of the simulator converges. The specialists verify the convergence of the error margin curve manually at present. After machine learning they verify the judgment for each role, and if the verification results are not good, they return to P2 and revise the metrics.

B. Detection Phase

P4. Role Candidate Judgment

Developers input programs to be detected into the metrics measurement system, and obtain measurements for each class. And developers input these measurements to the machine learning simulator. This simulator outputs values between 0–1 to all roles to be judged. The output values are normalized such that the sum of all values becomes 1. These output values are called role agreement values. A larger role agreement value means that the role candidate is more likely correct. The reciprocal of the number of roles to be detected is set as a threshold, and the role agreement values that are higher than the threshold are taken to be role candidates. The threshold is 1/12 (i.e., 0.0834) because we treat 12 roles at present. The sum of the output values is different at each input in the neural net work. Therefore, to use a common threshold for all the output, our technique normalizes the output value.

For example, Figure 8 shows the role candidate judgment results with NOM of 3 and NOAM of 2 and other metrics of 0; the output value of `AbstractClass` is the highest value. By regularizing the values of Figure 8, the roles are judged to be `AbstractClass` and `Target`.

P5. Pattern Detection

Developers input role candidates judged in P4 into the pattern detection system by using the pattern structure definitions defined in P1. And, this system detects patterns by matching the direction of the relations between role candidates and the roles of pattern in programs. The matching moves sequentially from the role candidate with the highest agreement value to that with the lowest value. The pattern detection system searches all combinations of role candidates that accord with the pattern structures. The pattern detection system detects patterns when the directions of relations between role candidates accord with the pattern structure and when the role candidates accord with roles at both ends of the relations. Moreover, the relation agreement values reflect the kind of relation.

Currently, our method deals with inheritance, interface implementation, and aggregation relations. The kind of relations will increase as more patterns get added in the future. The relation agreement value is 1.0 when the kind agrees with the relation of the pattern, and it is 0.5 when the kind does not agree. If the relation agreement value is 0 then the kind of relation does not agree, the pattern agreement value might become 0, and these classes will not be detected as patterns. In such cases, a problem similar to those of the previous detection techniques will occur because the difference in the kind of relation is not recognized.

The pattern agreement value is calculated from the role agreement values and the relation agreement values. The pattern to be detected is denoted as *P*, the role set that composes the pattern is denoted as *R*, and the relation set is denoted as *E*. Moreover, the program that is the target of detection is defined as *P'*, the set of classes comprising the role candidates is *R'*, and the set of relations between elements of *R'* is denoted as *E'*. The role agreement value is denoted as *Role*, and the relation agreement is denoted as *Rel*. *Role* means the function which is input the element of *R* and the one of *R'* , and . *Rel* means the function which is input the element of *E* and the one of *E'*. The product of the average of two roles at both ends of the relation and *Rel* is denoted as *Com*, and the average of *Com* is denoted as *Pat*. Moreover, the average of two *Role*s is calculated when *Com* is calculated, and the average value of *Com* is calculated to adjust *Pat* and *Role* to values from 0 to 1 when *Pat* is calculated. If the directions of the relations do not agree, *Rel* is assumed to be 0.

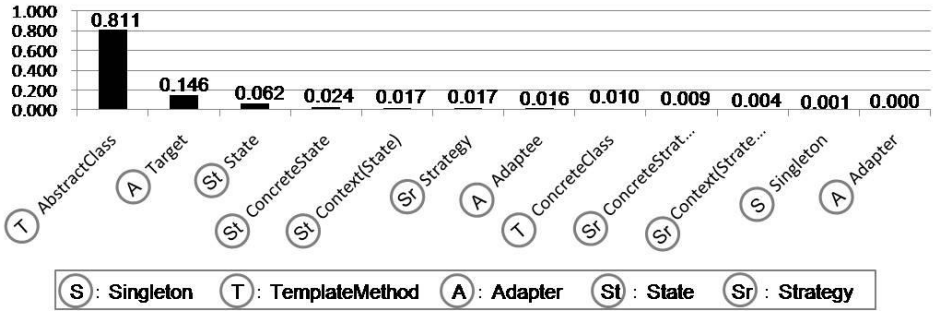


Figure 8. Example of machine learning output

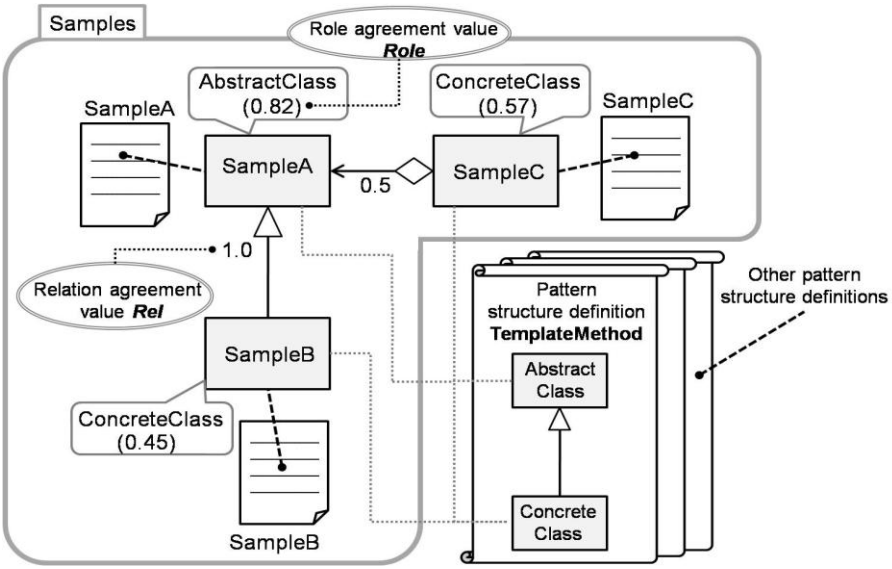


Figure 9. Example of pattern detection (TemplateMethod pattern)

$P = (R, E)$ $\quad P' = (R', E')$

$R = \{r_1, r_2, \ldots, r_i\}$ $\quad R' = \{r'_1, r'_2, \ldots, r'_k\}$

$E = \{e_1, e_2, \ldots, e_j\} \subseteq R \times R$ $\quad E' = \{e'_1, e'_2, \ldots, e'_l\} \subseteq R' \times R'$

$Role(r_m, r'_n) =$ The role agreement value $\quad r_m \in R, r'_n \in R'$

$Rel(e_p, e'_q) =$ The relation agreement value $\quad e_p \in E, e'_q \in E'$

$$Com(e_p, e'_q) = \frac{Role(r_a, r'_b) + Role(r_c, r'_d)}{2} \times Rel(e_p, e'_q)$$

$$r_a, r_c \in R, r'_b, r'_d \in R', e_p = (r_a, r_c), e'_p = (r'_b, r'_d)$$

$$Pat(P, P') = \frac{1}{\left|\left\{(e_p, e'_q) \in E \times E' \,\middle|\, Rel(e_p, e'_q) > 0\right\}\right|} \sum_{e_p \in E, e'_q \in E'} Com(e_p, e'_q)$$

Figure 9 shows an example of detecting the `TemplateMethod` pattern. In this example, it is assumed that class `SampleA` has the highest role agreement value for an `AbstractClass`. The pattern agreement value between the program Samples and the `TemplateMethod` pattern is calculated with the following algorithm.

$P = TemplateMethod = (R, E)$

$R = \{AbstractClass, ConcreteClass\}$

$E = \{AbstractClass \lhd - ConcreteClass\}$

$Samples = (R', E')$

$R' = \{SampleA, SampleB, SampleC\}$

$E' = \{SampleA \lhd - SampleB, SampleA \leftarrow\!\diamond SampleC\}$

($\lhd -$: inheritance, $\leftarrow\!\diamond$: aggregation)

$Role(AbstractClass, SampleA) = 0.82$ $\quad Role(ConcreteClass, SampleB) = 0.45$

$Role(ConcreteClass, SampleC) = 0.57$

$Rel(AbstractClass \lhd - ConcreteClass, SampleA \lhd - SampleB) = 1.0$

$Rel(AbstractClass \lhd - ConcreteClass, SampleA \leftarrow\!\diamond SampleC) = 0.5$

$$Com(AbstractClass \lhd - ConcreteClass, SampleA \lhd - SampleB) = \frac{0.82 + 0.45}{2} \times 1.0 = 0.635$$

$$Com(AbstractClass \lhd - ConcreteClass, SampleA \leftarrow\!\diamond SampleC) = \frac{0.82 + 0.57}{2} \times 0.5 = 0.348$$

$$Pat(TemplateMethod, Samples) = \frac{1}{2} \times (0.635 + 0.348) = 0.492$$

In the program shown in Figure 9, the pattern agreement value of the `TemplateMethod` pattern was calculated to be 0.492. Pattern agreement values are normalized from 0 to 1, just like role agreement values. Our technique uses the same threshold among of pattern agreement value as role agreement value because a lot of classes are detected as the pattern that composed of the only class like the singleton pattern. Classes with a pattern agreement value that exceeds the threshold are output as the detection result. The reciprocal of the number of roles for detection is taken to be

the threshold, similar to the case of role candidate judgment, and pattern agreement values that are higher than the threshold are output as the detection result.

In Figure 9, `SampleA`, `SampleB`, and `SampleC` were detected as `TemplateMethod` patterns. Moreover, `SampleA` and `SampleB`, `SampleA` and `SampleC` can also be considered to match the `TemplateMethod` pattern. In this case, the relation of "`SampleA` ◁– `SampleB`" is more similar to a `TemplateMethod` pattern than the relation of "`SampleA` ←◇ `SampleC`" because its agreement value of the former pair is 0.635 while that of the latter pair is only 0.348.

IV. Evaluation and Discussion

We determined whether the machine learning simulator derived identifying elements of the roles after learning. Moreover, we compared our technique with two previous techniques to verify the precision and recall of our approach and to confirm whether it could match its detected patterns with similar structures and diverse patterns.

A. Verification of Role Candidate Judgement

We used cross-validation to verify the role candidate judgment. In cross-validation, data are divided into n groups, and a test to verify a role candidate judgment is executed such that the testing data are one data group and the learning data are n-1 data groups. We executed the test five times by dividing the data into five groups. In this paper, programs such as programs in the reference [18], etc., are called small scale, whereas programs in practical use are called large scale. We used the set of programs where patterns are applied in small-scale programs (60 in total) [1] [18][19] and large-scale programs (158 in total from the Java library 1.6.0_13 [20], JUnit 4.5 [21], and Spring Framework 2.5 RC2 [22]) as experimental data. We judged manually and qualitatively whether the patterns were appropriately applied in this set of programs.

Table II shows the metrics that were chosen for the small-scale and large-scale programs. We used different metrics depending on the magnitude of the programs. For instance, we chose the metric called number of methods generating instance (NMGI) for small-scale programs because the method for transit states in the `ConcreteState` role in the `State` pattern generates other `ConcreteState` roles in small-scale programs. We guessed that the difference appeared in ratios of metrics about State and Strategy, so we used the same metrics for the large-scale programs without NMGI. Because State pattern treats the states in State role and treats the actions of the states in the Context role. On the other hand Strategy pattern encapsulates the processing of each algorithm into a Strategy role, and Context processing becomes simpler compared with that of State pattern.

[1] All small-scale code:

http://www.washi.cs.waseda.ac.jp/ja/paper/uchiyama/dp.html

We focused our attention on recall because the purpose of our technique was detection covering diverse pattern applications. Recall indicates the degree to which detection results are free of leakage, whereas precision shows how free of disagreement these result are. The data in Table III was used to calculate recall. w_r, x_r, y_r, and z_r are numbers of roles, and w_p, x_p, y_p, and z_p are numbers of patterns. Recall was calculated from the data in Table III by the following expressions.

Recall of role candidate judgment: $\mathrm{Re}_r = \frac{w_r}{w_r + x_r}$

Table IV shows the average recall for each role. Role candidates must be judged accurately because the `State` pattern and `Strategy` pattern have the same class structure. Therefore, our technique regards the roles of the patterns other than `State` and `Strategy` patterns as role candidates when the role agreement value was above the threshold, whereas our technique regards the roles of `State` and `Strategy` patterns as role candidates when the role agreement value was above the threshold and the roles of both patterns were distinguished State pattern from Strategy pattern.

As shown in Table IV, the recalls for the large-scale programs were lower than those for the small-scale programs. Accurate judgment of large-scale programs was more difficult because these programs possessed attributes and operations that were unnecessary for pattern composition. Therefore, it will be necessary to collect a significant amount of learning data to adequately cover a variety of large-scale programs.

The results in Table IV pertain to instances where the `State` and `Strategy` patterns could be distinguished. The `Context` role had high recall, but `State` and `ConcreteState` roles had especially low recalls for large-scale programs. However, the candidates for the `State` role were output with high recall when the threshold was exceeded. Therefore, the `State` pattern can be distinguished by initiating searching from the `Context` role in P5, and this improves recall.

TABLE II. Chosen Metrics

Abbreviation	Content
NOF	Number of fields
NSF	Number of static fields
NOM	Number of methods
NSM	Number of static methods
NOI	Number of interfaces
NOAM	Number of abstract methods
NORM	Number of overridden methods
NOPC	Number of private constructors
NOTC	Number of constructors with argument of object type
NOOF	Number of object fields
NCOF	Number of other classes with field of own type
NMGI	Number of methods to generate instances

TABLE III. INTERSECTION PROCESSION

	detected	not detected
correct	w_r, w_p (true positive)	x_r, x_p (false negative)
incorrect	y_r, y_p (false positive)	z_r, z_p (true negative)

TABLE IV. RECALL OF CANDIDATE ROLE JUDGMENT (AVERAGE)

		Average recall (%)	
Pattern	Role	Small-scale programs	Large-scale programs
Singleton	Singleton	100.0	84.7
Template Method	AbstractClass	100.0	88.6
	ConcreteClass	100.0	58.5
Adapter	Target	90.0	75.2
	Adapter	100.0	66.7
	Adaptee	90.0	60.9
State	Context	60.0	70.0
	State	60.0	46.7
	ConcreteState	82.0	46.6
Strategy	Context	80.0	55.3
	Strategy	100.0	76.7
	ConcreteStrategy	100.0	72.4

B. Pattern Detection Results

Our technique detects patterns using test data in both the small-scale and large-scale programs, and this result is evaluated. We used 40 sets of programs where patterns are applied in small-scale programs and 126 sets of programs where patterns are applied in large-scale programs as learning data. We judged manually and qualitatively whether the patterns were appropriately applied in the detection results. Table V shows precision and recall of the detected patterns. Precision and recall were calculated from the data in Table III by the following expressions:

$$\text{Recall of pattern detection :} \quad \mathrm{Re}_p = \frac{w_p}{w_p + x_p}$$

$$\text{Precision of pattern detection :} \quad \mathrm{Pr}_p = \frac{w_p}{w_p + y_p}$$

Small-scale and large-scale programs shared a common point in that they both had recalls that were higher than precisions. However, there were many non-agreements about the `State` patterns and `Strategy` patterns in the large-scale programs. Recall was 90% or more with the small-scale programs, but it dropped as low as 60% with the large-scale programs.

The large-scale programs resulted in especially low recall for the `Adapter` pattern. Table IV shows the cause: The recall of the role candidate judgment for the `Adapter` pattern was not high enough. It is necessary to show that all roles that compose patterns have agreement values that are above the threshold so that patterns will be detected. There were many cases in which neither of the roles that composed patterns was judged as a role candidate for the `Adapter` pattern. It will be necessary to return to P2 and choose new metrics. The `State` pattern was identified by searching from the `Context` role, for instance, in the `State` pattern detection in the large-scale programs, and the recall of the pattern detection was higher than the recall of role candidate judgment. Table V shows holistically that our technique suppresses false negatives because the recall is high.

TABLE V. PRECISION AND RECALL RATIO OF PATTERN DETECTION

	Number of test data		Precision (%)		Recall (%)	
Pattern	Small-scale programs	Large-scale programs	Small-scale programs	Large-scale programs	Small-scale programs	Large-scale programs
Singleton	6	6	60.0	63.6	100.0	100.0
Template Method	6	7	85.7	71.4	100.0	83.3
Adapter	4	7	100.0	100.0	90.0	60.0
State	2	6	50.0	40.0	100.0	66.6
Strategy	2	6	66.7	30.8	100.0	80.0

C. Experiment Comparing Previous Detection Techniques

We experimentally compared our technique with previous detection techniques [3][14]. These previous techniques have been publicly released, and they consider three or more patterns addressed by our own technique. Both target Java programs, as does our own. The technique proposed by Tsantails's technique [3](hereafter, TSAN) has four patterns in common with ours (`Singleton`, `TemplateMethod`, `Adapter` and `State`/`Strategy`). Because this technique cannot distinguish the `State` pattern from the `Strategy` pattern, these are detected as one pattern. Dietrich's technique [14] (hereafter, DIET) has three patterns in common (`Singleton`, `TemplateMethod`, `Adapter`) with our own. TSAN detects patterns based on the degree of similarity between the graphs of the pattern structure and graphs of the programs to be detected, whereas DIET detects patterns by using formal OWL (Web Ontology Language) definitions. Patterns were detected and evaluated with the small-scale and large-scale test data. Moreover, the test data and learning data were different.

Figure 10 shows the recall and precision graphs for our technique and TSAN, and Figure 11 shows the corresponding graphs for our technique and DIET. We ranked the detection results of our technique with the pattern agreement values. Next, we calculated recall and precision according to the ranking and plotted them. Recall and precision were calculated from the data in Table III by using the expressions of paragraph IV -*B*. In the results of TSAN and DIET, we alternately plotted results because these previous detection techniques output no value to rank. In the recall and precision graphs higher values are better.

Figure 10 and 11 show particularly good results for all techniques when small-scale programs was examined. This is because small-scale programs do not include unnecessary attributes and operations in the composition of patterns.

Table VI and VII show the average F measure for each plot of Figure 10 and 11. The F measure is calculated with

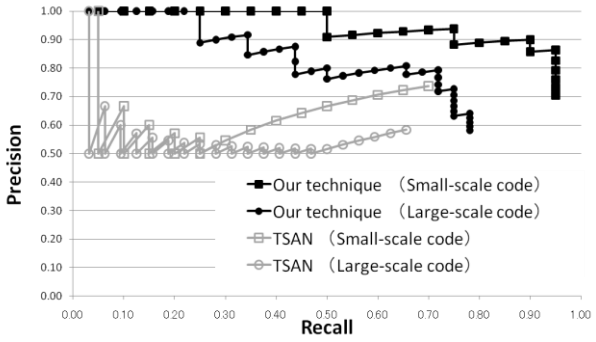


Figure 10. Recall-precision graph of detection results (vs. TSAN)

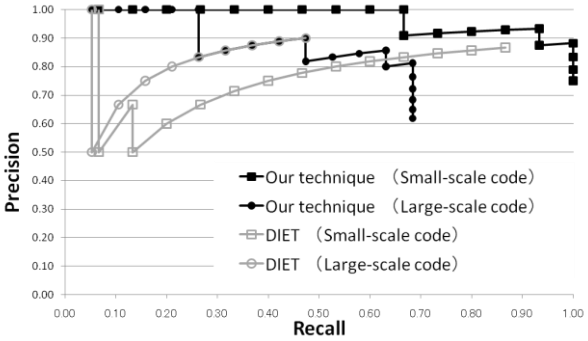


Figure 11. Recall–precision graph of detection results (vs. DIET)

TABLE VI. THE AVERAGE OF F MEASURE (VS. TSAN)

	Small-scale programs	Large-scale programs
Our technique	0.67	0.56
Previous technique (TSAN)	0.39	0.36

TABLE VII. THE AVERAGE OF F MEASURE (VS. DIET)

	Small-scale programs	Large-scale programs
Our technique	0.69	0.55
Previous technique (DIET)	0.50	0.35

recall and precision calculated by the above-mentioned expression as follows.

$$F-measure = \frac{2 \cdot \mathrm{Pr}_p \cdot \mathrm{Re}_p}{\mathrm{Pr}_p + \mathrm{Re}_p}$$

A large F measure means higher accuracy, and these tables show that our technique had a larger F measure than the previous techniques had.

Distinction between State *pattern and* Strategy *pattern*

Our technique distinguished `State` pattern `Strategy` pattern. Table VIII is an excerpt of the metrics measurements for the Context role in `State` pattern and `Strategy` pattern that were distinguished by the experiment on the large-scale programs. `State` pattern treats the states in `State` role and treats the actions of the states in the `Context` role. `Strategy` pattern encapsulates the processing of each algorithm into a `Strategy` role, and `Context` processing becomes simpler compared with that of `State` pattern. Table VIII shows 45 fields and 204 methods as the largest in `Context` role in `State` pattern (18 and 31 respectively in `Context` role of `Strategy` pattern). Therefore, the complexity of `Context` role of both patterns appears in the number of fields and the number of methods, and these are distinguishing elements. Figure 10 shows that our technique is especially good because `State` pattern and `Strategy` pattern could not be distinguished with TSAN.

Detection of Subspecies of Patterns

Figure 11 shows that the recall of DIET is low in the case of large-scale programs because this technique doesn't accommodate the diversity in pattern applications. Additionally, large-scale programs not only contain many attributes and operations in the composition of patterns but also subspecies of patterns.

Our technique detected subspecies of patterns. For example, our technique detected the source code of the `Singleton` pattern that used the boolean variable as shown in Figure 12. This `Singleton` pattern was not detected in TSAN or DIET. However, unlike the previous techniques, our technique is affected by false positives because it is a gradual detection using metrics and machine learning instead of strict conditions. False positives of the `Singleton` pattern especially stood out because `Singleton` pattern is composed of only one role. It will be necessary to use metrics that are specialized to one or a few roles to make judgments about patterns composed of one role like the `Singleton` pattern (P4).

Therefore, our technique is superior to previous one because the curve of our technique is above the previous in Figures 10 and 11.

TABLE VIII. MEASUREMENTS OF THE CONTEXT ROLE'S METRICS

Pattern - Role	Number of fields	Number of methods
State - Context	12	58
	45	204
	11	72
Strategy - Context	18	31
	3	16
	3	5

```
class Connection{
  public static boolean haveOne = false;
  public Connection() throws Exception{
    if (!haveOne) {
       haveOne = true;
    }else {
      throw new Exception(
      "cannot have a second instance");
    }
  }
  public static Connection getInstance()
  throws Exception{
  ...}
}
```

Figure 12. Example of diversity in pattern application (Singleton pattern)

V. CONCLUSION AND FUTURE WORK

We devised a pattern detection technique using metrics and machine learning. Role candidates are judged using machine learning that relies on measured metrics, and patterns are detected from the relations between classes. We worked on the problems associated with overlooking patterns and distinguishing patterns in which the class structures are similar.

We demonstrated that our technique was superior to two previous detection techniques by experimentally distinguishing patterns in which the class structures are similar. Moreover, subspecies of patterns were detected, so we could deal with a very diverse set of pattern applications. However, our technique was more susceptible to false positives because it does not use strict conditions such as those used by the previous techniques.

We have several goals for our future work. First, we plan to add more patterns to be detected. Our technique can currently cope with five patterns. However, we predict it will be possible to detect other patterns if we can decide upon metrics to identify them. It is also necessary to collect more learning data to cover the diversity in pattern applications. Moreover, we plan to more narrowly adapt the metrics to each role by returning to step P2 because results might depend on the data. This process would lead to the enhancement of recall and precision.

Second, we currently qualitatively and manually judge whether to return to step P2 and to apply GQM again; hence, in the future, we should find an appropriate automatic judgment method.

Third, we plan to prove the validity of the expressions and the parameters of agreement values and thresholds. We consider that it is possible to reduce the false positive rate by deciding on the optimum thresholds for role agreement values and pattern agreement values.

Finally, we plan to determine the learning number of times automatically and examine the correlation of the learning number of times and precision.

REFERENCES

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994.

[2] M. Lorenz and J. Kidd Object-Oriented Software Metrics. Prentice Hall, 1994.

[3] N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, and S. Halkidis. Design Pattern Detection Using Similarity Scoring. IEEE Trans. Software Engineering, Vol.32, No.11, pp. 896-909 2006.

[4] A. Blewitt, A. Bundy, and L. Stark. Automatic Verification of Design Patterns in Java. In Proceedings of the 20th International Conference on Automated Software Engineering, pp. 224–232, 2005.

[5] H. Kim and C. Boldyreff. A Method to Recover Design Patterns Using Software Product Metrics. In Proceedings of the 6th International Conference on Software Reuse: Advances in Software Reusability, pp. 318-335, 2000.

[6] R. Ferenc, A. Beszedes, L. Fulop, and J. Lele. Design Pattern Mining Enhanced by Machine Learning. 21st IEEE International Conference on Software Maintenance, pp. 295-304 2005.

[7] H. Washizaki, K. Fukaya, A. Kubo, and Y. Fukazawa. Detecting Design Patterns Using Source Code of Before Applying Design Patterns. 8th IEEE/ACIS International Conference on Computer and Information Science, pp. 933-938, 2009.

[8] N. Shi and R.A. Olsson. Reverse Engineering of Design Patterns from Java Source Code. 21st IEEE/ACM International Conference on Automated Software Engineering, pp. 123-134, 2006.

[9] H. Lee, H. Youn, and E. Lee. Automatic Detection of Design Pattern for Reverse Engineering. 5th ACIS International Conference on Software Engineering Research, Management and Applications, pp. 577-583, 2007.

[10] L. Wendehals and A. Orso. Recognizing Behavioral Patterns at Runtime Using Finite Automata. In 4th ICSE 2006 Workshop on Dynamic Analysis, pp. 33–40, 2006.

[11] S. Hayashi, J. Katada, R. Sakamoto, T. Kobayashi, and M. Saeki. Design Pattern Detection by Using Meta Patterns. IEICE Transactions, Vol. 91-D, No. 4, pp. 933–944, 2008.

[12] A. Lucia, V. Deufemia, C. Gravino and M. Risi. Design pattern recovery through visual language parsing and source code analysis. Journal of Systems and Software, Vol.82 (7), pp. 1177-1193, 2009.

[13] Y. Guéhéneuc and G. Antoniol. DeMIMA: A Multilayered Approach for Design Pattern Identification. IEEE Trans. Software Engineering. Vol.34, No. 5, pp. 667–684, 2008.

[14] J. Dietrich, C. Elgar. Towards a Web of Patterns. In Proceedings of First International Workshop Semantic Web Enabled Software Engineering, pp. 117-132, 2005.

[15] V. R. Basili and D.M. Weiss. A Methodology for Collecting Valid Software Engineering Data. IEEE Transactions on Software Engineering, Vol. 10, No. 6, pp. 728–738, 1984.

[16] T. Segaran. Programming Collective Intelligence. O'Reilly. 2007.

[17] H. Hirano. Neural network implemented with C++ and Java. Personal Media. 2008.

[18] H. Yuki. An introduction to design pattern to study by Java. http://www.hyuki.com/dp/

[19] H. Tanaka. Hello World with Java! http://www.hellohiro.com/pattern/

[20] Oracle Technology Network for Java Developers. http://www.oracle.com/technetwork/java/index.html

[21] JUnit.org. Resources for Test Driven Development. http://www.junit.org/

[22] SpringSource.org. Spring Source. http://www.springsource.org/

Using the Tropos Approach to Inform the UML Design: An Experiment Report

Andrea Capiluppi Cornelia Boldyreff
School of Computing, Information Technology and Engineering
University of East London
London, United Kingdom
{*a.capiluppi,c.boldyreff*}*@uel.ac.uk*

***Abstract*—Tropos is an agent-oriented software engineering (AOSE) methodology, based on the notion of actors, with goals and plans, and spanning all the phases of software development, from the very early phases of requirements analysis down to the actual implementation. The effectiveness of such methodology in the production of better design documents is evaluated in this study, by investigating the null hypothesis "using the Tropos Methodology before the analysis and design phases can produce a more accurate and complete set of UML diagrams than when no such technology is used".**

The evaluation of a real case scenario was given as part of a coursework in a BSc module at the University of East London, and the Tropos and UML diagrams were requested as part of the deliverables. The results of how students performed in such tasks, and how the Tropos approach helped in the drawing of the UML diagrams, are presented here.

The results show that generally, and confined to this experiment, the Tropos methodology has not helped in the design of the UML diagrams, and that students failed in understanding the link between the two methodologies.

***Keywords*-Software Quality; UML; Tropos Methodology**

I. Introduction

Among the core skills employed during the phase of requirements gathering and elicitation, is that of being able to identify and model the basic concepts of the application domain upon which the software system will be built. Such activity has been named *conceptual modelling*, and it serves the purpose of glueing together the requests by the customers, and the expertise of designers and developers, providing a platform to ease communication, meet users expectations and distribute knowledge [1]. Two techniques have recently been considered and compared for the modeling of such concepts, one based on scenarios of *how* the system is going to behave (or how the users will interact with it (e.g., the UML approach [2]); the other expressing *what* are the needs that the built system will fulfill, relating the business goals of the stakeholders with the functional and non-functional requirements of such system. The latter has been termed *goal-based* approach, and the agent-oriented software engineering (AOSE) methods have been one the more developed branches of such approach in the requirements elicitation.

Among the goal-based approaches, Tropos is an AOSE methodology based on two key ideas: agents and their interactions within the system environment. The main aim of Tropos is to produce a better understanding of the application domain where a system will operate, and of the kind of interactions that should occur between such system and the human agents. Within Tropos, the notion of agent, together with their goals and plans, are used since the early analysis of requirements elicitation: in the early phase of such analysis, the organizational setting is studied for the purpose of better understanding the scenario. In the late phase of requirements gathering and elicitation, the system is also inserted in the operational environment as one actor: the dependencies with the other actors represent the system's functional and non-functional requirements.

In both phases, the *actor* and *goal* diagrams are produced as outcomes, with the system being inserted in the diagrams in the late phase, but not in the early phase. The actor diagram represents the overall view of all the actors with their high-level dependencies to other actors, while the goal diagram is a refinement of the former with emphasis on the goals of a single actor (see Figure 1).

The focus of this work is on the early and late phases of requirements elicitation covered by the Tropos methodology, where the business entities are identified as actors, their goals assessed, and their inter-dependencies defined. In the UML notation instead, as summarised in Table I, these two phases correspond to the production of a *model in the problem space* (MOPS [3]). Such a model comprises of a set of use cases and business class diagrams (i.e., diagrams documenting business entities, their attributes and behaviors). When the business entities are converted into implementable entities, the UML notation produces the Model of Solution Space (MOSS) with the aim of feeding such model to the design phase.

The aim of this paper is to compare the UML outcomes from the MOPS phase (use cases and business class diagrams) as produced by undergraduate and postgraduate students, when combining (or not) the Tropos methodology as a "treatment". The rationale of such experiments is to determine through evaluation whether the joint use of

	Early Requirements (ER)	**Late Requirements (LR)**
Tropos	ER actors and goals	LR actors and goals (with system)
UML	MOPS	MOPS + MOSS

Table I
TROPOS AND UML DELIVERABLES IN THE EARLY AND LATE PHASES OF REQUIREMENTS GATHERING

goal-based (Tropos) and scenario-based (UML) approaches should be preferred to the use of only a scenario-based approach in the production of quality UML diagrams.

This paper details one experiment where BSc students at the university of East London, UK, produced both Tropos and UML diagrams towards the assessment of a scenario where a software system has to be built. The UML and Tropos diagrams were assessed against the benchmark produced as a marking scheme, and it is questioned whether the presence of the Tropos methodology has helped in the completeness of the resulting UML diagrams. This paper is the first of two experiments, where the Tropos methodology is used to inform the UML design: we plan to replicate this experiment in the semester starting in February 2011, without the Tropos "treatment": students will be required to work on the same scenario, but no Tropos diagrams will be required (or taught), therefore allowing for the comparison of two different sets of UML diagrams. This will provide the basis for comparing the effectiveness (or not) of the two combined approaches.

II. BACKGROUND AND RELATED WORK

This paper builds upon the scenario-based and the goal-based approaches as two viable tools in the requirements elicitation phase and for validation purposes. As a practical exemplification of the scenario-based requirement engineering method, we have used the Jacobson's Use Case technique, which has been lately incorporated into the UML notation language [2]. Such a model is based on the notion of "scenario" which is a *sequence of interaction events between a system and its environment in the restricted context of achieving some implicit purposes* [4], [5].

On the other hand, this paper relies on the concepts of agents and the agent-oriented paradigm (AOSE), as one example of goal-oriented approach [6], [7], [8]. This second approach is based on agents interacting as a group within a system, not just reacting to stimuli, but also communicating, coordinating, and cooperating as an autonomous and social entity that can to achieve individual and organizational goals.

The main notations of UML (as a scenario-based methodology) and Tropos (as goal-based) are summarised in Figure 1 (taken from [5]). Specifically for the Tropos notation, every system can be thought of several actors, having goals to fulfill with the use of such system. Such goals could be "hard" or "soft", depending on whether it is clear what actions and plans (or resources) should be performed (or used) in order to achieve such goals. A Tropos "actor diagram" details the overall connections between all the actors in the scenario, where a dependee (e.g. actor_3 in Figure 1) fulfills the goal(s) of a depender (e.g., actor_1 in Figure 1). A Tropos "goal diagram" focuses more precisely on one actor, and tries and elaborates on what plans, actions and resources should be performed to achieve each goal, and which actors are needed to fulfill these goals.

In the literature, the effectiveness of goal-oriented and scenario-based approaches is analyzed in several works illustrating the application of different methods to case studies (e.g., [9], [10], [11] or comparing the strengths and limitations of the approaches according to different criteria (e.g., [12], [13]). However, to the best of our knowledge, experimental comparisons of these requirements modelling paradigms using different visualization methods are rare [5]. Such comparisons may raise insights and help decide which modelling paradigm to adopt for a given software development project. The "quality" of UML models, comprised guidelines for the aesthetic quality, have also been evaluated [14].

One important factor for comparison or evaluation is the immediacy in understanding the respective models by projects stakeholders, for instance by requirements analysts [15], who have to understand a given model during analysis and refinement tasks to accommodate new or changed requirements.

III. EMPIRICAL APPROACH

This section introduces the definitions used in the following empirical study and presents the general objective of this work, and it does that in the formal way proposed by the *Goal-Question-Metric* (GQM) framework [16]. The GQM approach evaluates whether a goal has been reached, by associating that goal with questions that explain it from an operational point of view, and providing the basis for applying metrics to answer these questions. This study follows this approach by developing, from the wider goal of this research, the necessary questions to address the goal and then determining the metrics necessary for answering the questions.

Goal: The long term goal of this research is to evaluate whether the Tropos methodology (as an experimental "treatment"), jointly with the UML MOSS notation, produce higher standards of conceptual modelling than the UML notation alone.

Question: In this paper, and considering a given scenario as a case study, the following research questions will be evaluated:

1) Are the models produced by the students with the Tropos notation "complete" against a given benchmark?

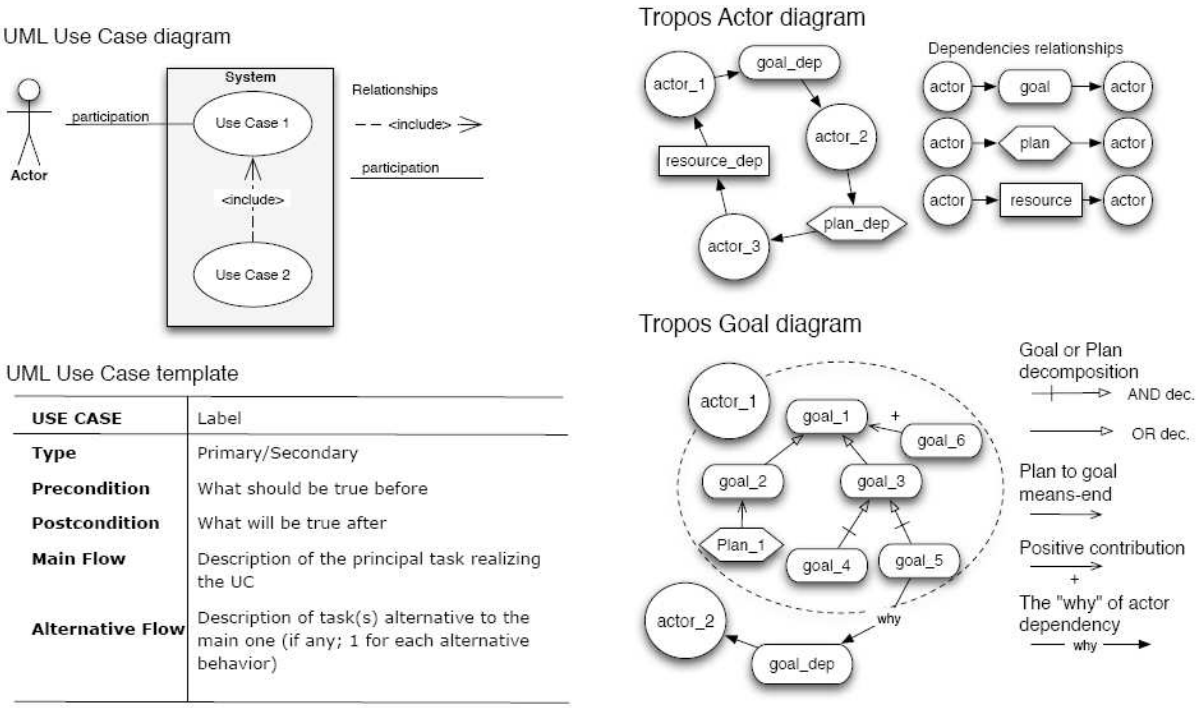


Figure 1. Main UML and Tropos concepts and notations (from [5])

Rationale: the aim of this question is to check whether the diagrams produced with the Tropos notation are compliant with a minimum list of actors and goals directly derived from the scenario. Such list of actors and goals should be considered as the "absolutely mandatory" in a typical requirements elicitation and gathering phase.

2) Are the models produced with UML complete against a given benchmark?
   Rationale: similarly to the question above, the aim of this question is to check whether the diagrams produced with a UML editor (Rational Rose, ArgoUML, etc) can be mapped to a minimum list of use case diagrams, necessary to describe the how the users of the system interact with its functionalities.
3) Can students map the Tropos actors and goals to UML use cases?
   Rationale: the aim of this question is to evaluate whether the use of "goals" and "actors" can help in focusing on the main functionalities of the system, expressed as UML use cases. Given the set of Tropos diagrams produced by any group of students, and a benchmark mapping of "Goals-to-use-cases" (see last column of Table III), it will be evaluated how the Tropos diagrams have informed the specified group of students in the creation of use cases.

Metrics: The Tropos actor and goal diagrams for this scenario have been listed in their minimum form, i.e., the minimum number of functionalities that are expected for (and from) this system, corresponding to both functional and non-functional requirements (see Table II). Also, the minimum set of UML use cases has been developed and it served as a benchmark to evaluate how students assessed the scenario (see Table III). Each group coursework was evaluated against these two lists, and the number of correct diagrams produced by the students evaluated against these baselines.

IV. Experimental Design

The first part of the experiment was set up at the University of East London, during the Level 3 module "Advanced Information Systems Development". The experiment population comprised some 65 students, divided in 17 groups of 3 to 4 members[1]. Each group was in charge of producing two sets of diagrams: the Tropos goal and actor diagrams (for both the early and late phase of the requirements); and the UML use cases and class diagrams. All the students in the module had already studied the basic UML concepts in a previous module, while the Tropos concepts were introduced during several lectures, and their practical implementation was assisted in the lab sessions. The scenario was distributed to students on week 4 (out of 12 weeks in the module), and it represents the coursework needed to pass the module,

[1]Since the selection of students and groups was not random, the study should be referred to as a *quasi-experiment*. We will use the term "experiment" as a synonym throughout the study

together with the final exam. The students had 9 weeks to complete the task.

In order to produce the Tropos diagrams, the OME tool, implementing the i* notation[2], was taught and demonstrated during the lab sessions. In order to produce the UML class and use case diagrams, students could select the UML editor of their choice (e.g., the IBM Rational Rose toolkit, or the Open Source ArgoUML tool[3], etc.).

A. Scenario

The following problem statement was provided to the students, with the request for modeling such scenario via a scenario-based approach and a goal-based approach. This is based on a previous job placement where a student effectively designed and developed the system outlined below.

> *A company has supplied and supported its clients in the area of Tax and Returns Automation for more than 10 years. This involves an employee going to the client sites and inspecting the revenues that each of the client companies claim in a given year and giving advices and filling the necessary forms for Tax Return purposes. Once the employee has filled the relevant forms (on a per-client basis), these forms need to be saved to a couple of paper copies, one to be kept by the client, one to be archived within the company.*
> *The company is seeking to streamline and automatise its systems for record keeping, therefore enabling the business to offer their clients a better service. The aim of this project would be to develop a system allowing data collection during site visits to be entered onto an online application, that sits on the web: the employee visiting the site's premises would input the data to a specified form (which can be extended by a System Administrator to contain more fields and input data, it could be reused from existing form, and new forms can be created ad-hoc). The data once collected would be synchronized with the companys database, but during the initial trial period, the paper-based system, and the on-line system, would need to run together, and be synchronised.*
> *The data collected would be used to keep the clients informed of the results of the employee's visits and the next visit's date. This upgrade project would be expected to cover the following areas: data acquisition using online, secure systems; synchronizing of data; a database to store the data of clients; and a PC based management tool for the data-captured database.*

[2]*OME3*, available online at http://www.cs.toronto.edu/km/ome/
[3]ArgoUML, freely available at http://argoUML.tigris.org/

B. Expected Outcomes – Tropos Marking Scheme

In order to assess the courseworks produced by the students, a list of "model solutions" was produced, and checked against the delivered set of diagrams. In particular, a minimum list of the Tropos actors present in the scenario was produced and their main goals were identified: the following Table II was therefore used as the baseline for marking the assignment. These goals and actors were prepared by one of the authors (running the module) and the assistant, a PhD student whose focus is on the secure aspects of Tropos.

Three main actors (Client, Company and Employee) were identified as expressing goals within the interaction with the system, while other two (the System, and the HM Revenue and Customs agency – HMRC) are also present, acting as dependees in one or more of those goals by the three main actors.

The marks available for the completion of such task were 25 out of 50.

Goal-based approach – TROPOS			
Actor	Goals – (H)ard or (S)oft		Dependee
Company	GCo1	Schedule periodic meetings (H)	Client
	GCo2	Get data to fill forms (H)	Client
	GCo3	Get up-to-date Returns rules (H)	HMRC
	GCo4	Secure data based on client or employee (H)	System
	GCo5	Provide a better service to clients (S)	Self
	GCo6	Rationalise forms (S)	Self
Employee	GE1	Get training on up-to-date procedures (H)	Company
	GE2	Get online access during visits (H)	Client
	GE3	Access clients details on system (H)	System
	GE4	Log activity or duration (H)	System
Client	GCl1	Obtain copies of job performed (H)	Employee
	GCl2	Get Tax Return advices (H)	Employee
	GCl3	Browse activity logs (H)	System
	GCl4	Get secure service (S)	Company
System			
HMRC			

Table II
MARKING SCHEME – TROPOS ACTORS AND GOALS

C. Expected Outcomes – UML Marking Scheme

The following Table III summarises instead the list of UML use cases that were set up as a baseline for marking the scenario-based part of the assignments: three main UML actors were expected to be interacting with the system, with increasing amount and type of provileges: the clients of

the Tax Revenue company (c_i in Table III, $i = 1..5$), its employees (e_i in the same Table, with $i = 1..7$) and the system administrator (s_i in the same Table, with $i = 1..6$).

The UML use cases listed, and intended as a "model solution", are a subset of what was documented during a business consultancy,where the described system was actually implemented by a student in a job placement. The listed UML use cases should be inferred by reading the problem statement of the scenario, and they should also become clearer after working on the Tropos goals and actors. Albeit more specified UML actors could be identified (e.g., the ISP administrator, the project manager in charge of delivering the requested system, the Tax Revenue company owner, etc.), the above three provide the minimum set of scenarios that fulfill most of the functional and non-functional requirements of the scenario. In some of these, one UML use case could be the extension, or being included in some other use case (for instance, the "log-in" use case is typically included in any interaction with the system, independently from the privileges).

The marks available for the completion of the UML task were also 25 out of 50. This was decided to balance the relative importance of both Tropos and UML tasks.

D. Results

As said above, the results obtained from the marking of the presented coursework represent the first part of this study: the second part will be based on assessing the diagrams produced by the students when the "treatment" Tropos is not taught or requested.

Table IV shows how each group coursework (G1 to G17) was evaluated against the list of Tropos goals and UML use cases, gathered around the main actors expressing their requirements, either in a goal-based approach, or a scenario-based approach.

At a first glance, the results found in the table show that the students found easier to assess the Tropos actors and goals, rather than producing the relative UML diagrams to describe how the actors are interacting with the system. Even when breaking down the aggregated results in the main components and actors, it is visible that some actors were assessed better than others: the Tropos models for the Company providing the Tax Revenue service are more complete than other actors (as visible in Table V where on average 70% of the groups assessed the benchmark goals from the Company actor).

The striking difference with such a finding is visible by observing the results of the UML cases, summarised in Table VI, where on average, only 38% of the groups assessed the "Employee" use cases, and only 43% delivered the "administrator" cases. As a grand average, some 64% of groups successfully assessed the set of Tropos goals proposed as a baseline, while only 38% of students assessed the set of UML cases of the benchmark.

Tropos Goal	Groups delivering	Perc	Average
GCo1	12	70.59%	70.59%
GCo2	14	82.35%	
GCo3	8	47.06%	
GCo4	11	64.71%	
GCo5	17	100.00%	
GCo6	10	58.82%	
GE1	7	41.18%	58.82%
GE2	15	88.24%	
GE3	13	76.47%	
GE4	5	29.41%	
GCl1	12	70.59%	58.82%
GCl2	14	82.35%	
GCl3	8	47.06%	
GCl4	6	35.29%	
Grand Average			63.87%

Table V
RESULTS – BY GROUP

UML use case	Groups delivering	Perc	Average
c1	9	52.94%	27.06%
c2	3	17.65%	
c3	4	23.53%	
c4	3	17.65%	
c5	4	23.53%	
e1	7	41.18%	38.66%
e2	11	64.71%	
e3	5	29.41%	
e4	15	88.24%	
e5	2	11.76%	
e6	2	11.76%	
e7	4	23.53%	
a1	12	70.59%	43.53%
a2	3	17.65%	
a3	13	76.47%	
a4	5	29.41%	
a5	1	5.88%	
a6	15	88.24%	
Grand Average			38.56%

Table VI
RESULTS – BY GROUP

These discrepancies are also visible when considering single students groups:

- among the Tropos goals, 4 goals out of 6 were on average correctly identified, with regards to the Company goals (average 4.23 goals); among the Employee goals, 2 goals out of 4 were on average assessed (average 2.35 goals); finally, among the Client goals, 2 out of 4 goals were identified (average 2.35 goals);
- with respect to the UML cases, 1 out of 5 cases were identified for the client (average 1.35 cases); 2 out of 7

Scenario-based approach – UML			
UML actor	UML use cases		Via Tropos goal(s)
Client	c1	Can log-in	GCl3, GCl4, GCo4, GCo5
	c2	Can update their details	GCl4
	c3	Can browse the log of activity	GCL3, GE4
	c4	Can browse relevant documentation	GCo3, GCl4
	c5	Has sole access to private area	GCl4
Employee	e1	Can schedule visit	GCo1
	e2	Can log-in	GE2
	e3	Can select appropriate forms based on client	GE1, GCo6
	e4	Can fill in forms	GCo2, GE4
	e5	Can fill in the log of activity	GE4, GCl3
	e6	Can upload relevant documentation	GCo3
	e7	Has privileged access to all clients private area	GCo4, GCo5, GE3
Administrator	a1	Can log-in	GCo5
	a2	Can create/update/remove employees	GCo4, GE1
	a3	Can create/update/remove forms	GCo6
	a4	Can create/update/remove clients	GCo4
	a5	Can monitor the activity of employees	GE1, GE4
	a6	Can synchronise the database	GE1, GE4, GCo6

Table III
MARKING SCHEME – LIST OF UML USE CASES

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	G12	G13	G14	G15	G16	G17
GCo1	√	√	√		√	√	√				√	√	√		√	√	√
GCo2	√	√	√	√	√	√	√	√	√		√	√		√		√	√
GCo3		√	√	√			√				√	√	√		√		
GCo4		√	√	√	√		√	√			√	√	√			√	√
GCo5	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
GCo6		√		√	√		√	√				√	√		√	√	√
GE1		√		√			√				√	√			√	√	
GE2	√		√	√	√	√	√	√	√		√	√	√	√	√	√	√
GE3	√	√		√	√	√	√	√	√		√		√		√	√	√
GE4		√				√	√	√								√	
GCl1		√	√	√	√	√	√				√	√	√		√	√	√
GCl2	√	√		√	√	√	√		√	√	√	√	√		√	√	√
GCl3	√		√		√		√	√			√					√	√
GCl4			√		√			√		√	√				√		
c1	√	√						√		√	√	√			√	√	√
c2								√			√				√		
c3	√				√											√	√
c4					√											√	√
c5		√								√		√				√	
e1		√		√	√	√				√		√	√				
e2	√	√	√		√		√		√	√	√	√			√	√	
e3	√		√							√			√	√			
e4	√	√	√	√	√	√	√	√	√	√	√	√	√	√		√	
e5									√	√							
e6		√							√								
e7	√							√		√	√						
a1	√	√	√	√		√	√				√	√		√	√	√	√
a2								√	√			√					
a3	√	√			√	√	√	√			√	√	√	√	√	√	√
a4		√		√			√		√			√					
a5		√															
a6	√	√	√	√	√	√	√	√	√		√	√		√	√	√	√

Table IV
RESULTS – BY GROUP

cases were identified for the employee (average 2.70 cases); and 2 out of 6 cases were assessed for the administrator of the system (average 2.88 cases)

Relatively to the experiment performed with the students of the University of East London, we can conclude that the use of the Tropos approach was not effective to inform the UML conceptual model.

V. Threats to Validity

Like any other empirical study, the validity of ours is subject to several threats. In the following, threats to *internal* (whether confounding factors can influence your findings), *external* (whether results can be generalized), and *construct validity* (relationship between theory and observation) are illustrated.

- **Internal validity** – the terminology "quality of UML models" was used to define whether "better" models could be obtained with the use of the additional Tropos analysis. Obviously the quality of UML diagrams is a multi-faceted dimension of several possible: aesthetic aspects could be considered, but also others based on design metrics of UML diagrams, as coupling, complexity, etc).
- **External validity** – the following threats to external validity have been identified:
  1) these findings cannot be generalised by one scenario, distributed to some 70 students, and based on one observation only. Replications are needed not only regarding the presence or absence of the Tropos "treatment", but also with more students involved.
  2) Despite the initial results, a stronger statistical formalism cannot be used for investigating the research questions: this is because since there is no comparison with a null hypothesis, such analysis cannot be properly performed. The results will become much more reliable when the second part of the experiment will be carried out.
- **Construct validity**: the minimum set of actor and goal diagrams, and the minimum set of UML use cases derived for the construction of the benchmark could play an important part in the outcomes of this experiment. The reason of choosing these use cases, and the relative Tropos actors and goals, are of a practical nature: the proposed one is a real scenario of a past job placement, where a student designed and implemented the system to be delivered: the "model answers" are a subset of the diagrams implemented for the deployment of such system.

VI. Acknowledgements

The authors would like to thank Dr H. Mouratidis and Michalis Pavlidis for the extensive comments, and the help in formulating the Tropos goals and actors. The authors would also like to thank the anonymous reviewers, since many improvements were added to the text, based on their suggestions.

VII. Conclusion and Further Work

The usage of visual modelling tools has become a common support for the design of a software system's capabilities; the use of such tools has become more valuable in the early phase of requirement gathering, where the interaction with non-technical stake-holders requires jargon-free and easily usable approaches. Among these techniques, this paper has considered the goal-based (Tropos) and the scenario-based (UML) methodologies, trying to assess whether the use of the first could be useful to inform the definition of more complete use cases.

A set of "model solutions" was prepared for a given scenario, that was handed out as part of a coursework at the University of East London, UK. A baseline set of actors was prepared for the Tropos approach, and one for the UML use cases. Each coursework was assessed against these two baselines. Contrarily to what was expected, a larger number of students correctly assessed a larger amount of Tropos goals, whereas the UML cases were delivered less often, and more erroneously. Although the correct UML cases were assessed where the relevant Tropos actors were identified, this was not always the case: students found it difficult to connect the two approaches, and synchronise the actors and goals with how the system was supposed to perform.

These results are interesting, but we need to produce a similar set of observations when removing the Tropos approach from the experiment: we plan to replicate this experiment in a course starting in February 2011, where the same scenario will be provided, and where only the UML use cases will be requested. This will help in assessing whether the use of the Tropos approach can be considered to play a difference in the requirements gathering phase, when coupled to the UML notation.

References

[1] A. van Lamsweerde, "Requirements engineering in the year 00: a research perspective," in *Proceedings of the* 22^{nd} *International Conference on Software engineering (ICSE 00)*. New York, NY, USA: ACM, 2000, pp. 5–19.

[2] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, 2^{nd} *Edition (Addison-Wesley Object Technology Series)*. Addison-Wesley Professional, 2005.

[3] B. Unhelkar, *Verification and Validation for Quality of UML 2.0 Models*. Wiley-Interscience, 2005.

[4] A. Sutcliffe, "Scenario-based requirements engineering," in *Proceedings of the* 11^{th} *IEEE International Conference on Requirements Engineering*. Washington, DC, USA: IEEE Computer Society, 2003, pp. 320–.

[5] I. Hadar, T. Kuflik, A. Perini, I. Reinhartz-Berger, F. Ricca, and A. Susi, "An empirical study of requirements model understanding: Use Case vs. Tropos models," in *Proceedings of the* 25^{th} *ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2010, pp. 2324–2329.

[6] J. Mylopoulos, "Information modeling in the time of the revolution," *Information Systems*, vol. 23, pp. 127–155, May 1998.

[7] E. S. Yu, "Modelling strategic relationships for process reengineering," Ph.D. dissertation, Toronto, Ont., Canada, Canada, 1996.

[8] A. Dardenne, A. van Lamsweerde, and S. Fickas, "Goal-directed requirements acquisition," *Science of Computer Programming*, vol. 20, pp. 3–50, April 1993.

[9] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini, "TROPOS: An agent-oriented software development methodology," *Autonomous Agents and Multi-Agent Systems*, vol. 8, no. 3, pp. 203–236, May 2004.

[10] J. Castro, M. Kolp, and J. Mylopoulos, "Towards requirements-driven information systems engineering: the tropos project," *Information Systems*, vol. 27, pp. 365–389, September 2002.

[11] M. Kim, S. Park, V. Sugumaran, and H. Yang, "Managing requirements conflicts in software product lines: A goal and scenario based approach," *Data & Knowledge Engineering*, vol. 61, pp. 417–432, June 2007.

[12] J. L.M.C. Filho, V.Werneck and E. Yu, "Agentgoal orientation vs object orientation for requirements engineering: A practical evaluation using an exemplar," in *Proceedings of the* 8^{th} *Workshop on Requirements Engineering*, 2005, pp. 123–134.

[13] C. Rolland, G. Grosz, and R. Kla, "Experience with goal-scenario coupling in requirements engineering," in *Proceedings of the* 4^{th} *IEEE International Symposium on Requirements Engineering*. Washington, DC, USA: IEEE Computer Society, 1999, pp. 74–.

[14] H. Eichelberger and K. Schmid, "Guidelines on the aesthetic quality of UML class diagrams," *Information and Software Technology*, vol. 51, no. 12, pp. 1686 – 1698, 2009.

[15] C. F. J. Lange, "Improving the quality of UML models in practice," in *Proceedings of the* 28^{th} *International Conference on Software engineering*. New York, NY, USA: ACM, 2006, pp. 993–996.

[16] V. R. Basili, G. Caldiera, and D. H. Rombach, "The goal question metric approach," in *Encyclopedia of Software Engineering*. John Wiley & Sons, 1994, pp. 528–532, see also http://sdqweb.ipd.uka.de/wiki/GQM.

Tool-Supported Estimation of Software Evolution Effort in Service-Oriented Systems

Johannes Stammel and Mircea Trifu
FZI Forschungszentrum Informatik
10-14 Haid-und-Neu Str., Karlsruhe, Germany
{stammel, mtrifu}@fzi.de

Abstract

Existing software systems need to evolve in order to keep up with changes in requirements, platforms and technologies. And because software evolution is a costly business, an early and accurate estimation of evolution efforts is highly desirable in any software development project. In this paper we present KAMP, a tool-supported approach, based on change impact analysis, enabling software architects to express a potential design for a given change request at the architecture level and to accurately estimate the evolution effort associated with its implementation in source code. The approach is also used to compare alternative designs before carrying out the implementation work. We apply the KAMP approach on an enterprise SOA showcase and show how it supports the predictable and cost-effective evolution of this software system.

Keywords: *software evolution, effort estimation, architecture, change impact analysis.*

1 Introduction

In order to keep service-oriented software systems up-to-date with changes in requirements, platforms and technologies they need to evolve. Evolution is very common and involves changes to the software system and its architecture. However, architecture changes may have significant impact on software quality attributes such as performance, reliability and maintainability, which is why it is highly desirable to be able to predict these quality impacts.

As a result, the topic of quality impact prediction for evolving service-oriented software systems has been addressed in various research initiatives, such as the Q-ImPrESS [3] research project. The project provides tool-supported methods that predict and analyze quality attributes on architecture level using formalized architecture models.

The project considers multiple quality attributes, i.e., performance, reliability and maintainability, that are conflicting to each other and provides support for balancing them and exploring trade-offs. This paper describes the maintainability prediction approach within the Q-ImPrESS project.

Within this paper, we focus on three types of evolution scenarios. The first type is the requirement of new or changed functionality, i.e., a new service needs to be implemented. The second type is a changed runtime environment, e.g., a new middleware platform, that has to be supported. And the third type is a changed usage profile, e.g., system has to cope with more users without impaired performance.

The implementation of an evolution scenario generates costs and for a good project management it is important to have control over costs and efforts related to evolution. For this reason within the context of the Q-ImPrESS project, we developed the KAMP (Karlsruhe Architectural Maintainability Prediction) approach, a tool-supported approach, based on change impact analysis, enabling software architects to express a potential design for a given change request at the architecture level. Its goal is to estimate the evolution effort associated with its implementation in source code.

The approach is based on architecture modeling using views for static structure, behavior and deployment of services. A service is represented as a deployed component providing service operations to its clients.

Note that evolution effort does not represent only implementation effort. KAMP is able to cover different efforts for different activities, utilizing the information annotated in the architecture models. In particular, KAMP considers management efforts for (re-)deployment and (re-)configuration.

Section 2 discusses foundations about estimation of software evolution efforts, maintainability and change requests. In Section 3 we present the KAMP approach. Section 4 describes the results of an initial case-study applying KAMP. Section 5 summarizes the related work, while section 6 concludes this paper.

2 Foundations

2.1 Effort Estimation

Nowadays, effort estimation is an essential part during planning and execution of software development projects. Most projects have a limited budget, therefore a careful planning is necessary to avoid running out of budget. Effort estimation helps to detect resource problems early and allows for timely corrections, where necessary.

Effort estimation approaches derive their estimation values from planned project scope and from anticipated complexity. In addition most approaches have plenty of input parameters.

Existing effort estimation approaches support different project phases. There are approaches, which focus on the requirements engineering phase, others aim at estimation during the design phase or the implementation phase.

The input parameters depend on the supported project phase and are determined by the project artifacts available at that point, i.e., during the requirement engineering phase the inputs come from the requirements document, during the design phase the inputs are based on design specification and during the implementation phase data and progress values of the running project can be used. The earlier the estimation is done the more imprecise the available data is and the less confident the estimations are.

So, in order to get valid results from estimation approaches the input parameters need to be calibrated to fit the given project context. This is done with data derived from similar projects for example by using a project database. However, such a database is not always available at the beginning. Nevertheless, the input parameters have to be readjusted while the project is progressing in order to fit the actual circumstances.

Therefore estimation should be established as a continuous task during the project life cycle. Since the architecture is one of the central artifacts for managing software development projects, effort estimation should be closely aligned with it, and support for seamless continuous effort estimation during the architecture design phase is highly desirable.

2.2 Maintainability Definition

Maintenance efforts represent a significant part of the total effort of a software development project. During the lifetime of a system, the system has to evolve in order to be still usable.

With respect to [12], we define maintainability as *"The capability of a software product to be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications"*. We focus our approach on the last part of the maintainability definition which is covering the evolution aspect.

Maintainability is strongly associated with the effort required to implement occurring change requests, which is why, for now, the KAMP approach is concerned with estimating maintenance efforts.

2.3 Change Requests

A change request is a particular situation when the software system needs to be modified. Since an architecture can not be arbitrarily flexible and implementing flexibility costs time and money, it is difficult to make a general statement about maintainability. Even patterns and anti-patterns are not clearly distinguishable on architecture level without relation to change requests. In order to reduce the effort for change requests that need to be implemented, one needs to anticipate which changes occur in the future. Overall our approach helps with estimating the efforts necessary for implementing anticipated change requests.

Within this paper we distinguish several *kinds of change requests*, based on their causes or stimuli. There are requirement changes regarding functionality, that request a new or altered functionality. Another stimulus is the evolution of the technical environment, which the software system depends on, e.g., changes in the platform (operating system, middleware). Another stimulus is the evolution of a COTS product, used by the software system, i.e., API changes of underlying libraries. Other stimuli arise from changed user profiles, (e.g., increased number of users, different usage behaviors), which require changes in order to fulfill non-functional aspects like performance, reliability, and security.

Besides the stimulus, the *effect of a change request* plays an important role in KAMP, since a change request needs to be translated into concrete work tasks. The effect is represented by all tasks and subtasks that lead to the fulfilment of the change request, including follow-up tasks due to change propagation. These tasks can affect various kinds of effort types. This comprises in the first place efforts for implementation (code changes), but also efforts for (re-)configuration, (re-)compilation, (re-)testing, (re-)deployment, data handling (modeling, conversion, migration), components-off-the-shelf (COTS) handling (survey, selection, tailoring, configuration, replacement), as well as efforts for retaining and increasing the internal code quality (refactoring, anti-pattern detection and removal).

3 The KAMP Approach

3.1 Overall

The Karlsruhe Architectural Maintainability Prediction (KAMP) approach aims to enable effort estimation for change requests based on architecture models.

Given a change request the approach derives from the architecture model a change description, i.e., work plan. This work plan contains the tasks, that are necessary for implementing the change request, coupled with tasks related to other activities like (re-)configuration, (re-)deployment of components, etc.

In order to get effort estimates, KAMP provides support to determine the complexity for each task in the work plan. A bottom-up estimation approach is used to map the complexity of each task to corresponding time effort. Overall KAMP combines a top-down derivation phase for creating the work plan with a bottom-up estimation phase. Note, in the current state of the approach the bottom-up estimates have to be provided manually by the user, whereas the workplan derivation is automated as is explained in the following sections.

The level of detail and granularity of the work plan tasks starts high, covering abstract tasks, and is then stepwise refined, by gathering additional information from the user and from architecture models, following a guided procedure. On the one hand the level of detail can be refined by going from the component level to the level of single service operations, while on the other hand the work plan description can be extended by following up on change activities that are detected using a semi-automated change impact analysis.

3.2 Inputs and outputs

KAMP takes as *inputs* 1) the description of the software architecture and 2) the description of the change request.

For the *description of the software architecture* the user creates an instance of Q-ImPrESS Service Architecture Meta-Model (SAMM) [2] that provides all elements of a component-based and service-oriented software architecture.

The architecture model can be created manually or, given certain conditions, retrieved automatically from source code by applying the Q-ImPrESS Reverse Engineering tool chain. A set of heuristics for detection of structural architecture parts, such as component boundaries and interfaces, as well as the statical analysable behaviour, is provided. In the project context of Q-ImPrESS these heuristics are applicable to Java and C/C++ code.

As an intermediate result of the reverse engineering, we obtain a Generalized Abstract Syntax Tree (GAST) and a mapping between the architecture model and the GAST. The GAST model can be used to calculate code and design metrics, which allows for an automatic determination of complexity metrics for corresponding architecture elements.

The description of a change request contains a name and an informal description of the stimulus, referring to the requirements that are affected by the change request. Moreover it covers the kind of stimulus (functional requirement change, technical environment change, COTS evolution, usage profile change).

The Q-ImPrESS SAMM allows for specifying alternative and sub-alternative models for various sequences of change requests, leading to a tree-like hierarchy of architectural models, each path within this hierarchy starting from the root model representing an evolution alternative of the software architecture. Each element in the tree represents a an architecture alternative, that consists of models for each supported architectural view, i.e., repository, system structure, behaviour, hardware environment, deployment, and quality annotations. Each subnode in the tree is basically a copy of its parent alternative with some modifications.

In order to specify how the change request is mapped to the architecture model, the user creates a sub-alternative of the actual system model and adapts the architecture model according to the change request.

The output of KAMP is a work plan, containing the change tasks, annotated with complexity values and effort estimates. Work plans can be compared, by comparing the structure or by comparing the aggregated complexity and effort values.

3.3 Work plan model

A work plan contains a list of activities or tasks. The types of activities are defined in a meta model. An activity refers to an architectural element and a basic activity.

The Q-ImPrESS Service Architecture Meta-Model (SAMM) [2] specifies the architecture elements like Component, Interface, Interface Port, Operation, Parameter, Datatype, etc.

Basic activities are **add**, **modify** and **remove**. **Add** means that the architecture element has to be newly implemented, **modify** means that the element has to be modified, and **remove** means that the element needs to be deleted from the code. For example the work plan can contain activities like ”Modify Component A”, ”Add Operation B”, or ”Remove Parameter C”.

Besides these implementation related work plan activities the work plan metamodel provides activities that cover other effort types related to configuration, testing and deployment. Provided activities are ”Modify configuration”, ”Run Tests”, ”Deploy components” and ”Update deployed

components".

3.4 Work plan derivation

The last section presented the ingredients of the work plan. Let us now have a look at how KAMP derives work plan instances. The work plan derivation in our approach is achieved in two ways. First, the work plan can be derived from changes in the architecture model, and second, the work plan can be derived by following a wizard dialog. The following paragraphs explain both ways in detail and compare them.

Derivation from architecture model changes The first way of work plan derivation is by calculating tasks from changes in the architecture model. In this way the user creates a copy of the architecture model and changes it according to the selected change request. KAMP calculates the differences from the changed architecture model to the base architecture model and translates the differences into work tasks. The work plan is filled with translated work tasks.

For example lets assume a client-server-database application. Now the software architect decides to introduce a cache between the server and the database. The architect creates a sub-alternative model and inserts the cache into the architecture model and fixes the interfaces and connectors using a model editor. Then, KAMP calculates the differences and creates a work plan containing an activity "Add Component Cache". Additionally the changes to interfaces and connectors are retrieved and represented by corresponding work plan activities.

Technically, the SAMM as well as the workplan metamodel are implemented using the EMF Ecore technology. Therefore we use EMFDiff to calculate a diff model between instances of SAMM. We defined a mapping between the elements of the diff model to corresponding workplan activities and wrote a transformation, that creates the workplan out of it.

Derivation by wizard dialog The second way of derivation is by following a wizard dialog. As the first step of this way the user determines the primary changes, i.e., architecture elements that need to be changed representing a starting point for the change. On the first wizard page the user marks the components that have to be added, modified or removed. On the second and third wizard pages the user refines this information to interface port and operation level, thus telling KAMP what interface ports or operations of the selected components need to be added, modified and removed.

For example, if a functionality in a user interface has to be modified, the user points out the components that build up the user interface and marks them with "modify". On the second and third wizard page the user marks the interface ports and operations of the user interface component that need to be modified.

Outgoing from this starting point the approach helps with identifying follow-up changes. For example, the components that are connected to the user interface components that implement the business logic of the functionality have to be changed due to the changes of the user interface components. Computing follow-up tasks is necessary because it ensures that all locations depending on a change, such as an interface signature or a behavior change, are changed consistently. If a change can not be kept locally it will propagate to other system parts.

The KAMP tool suite uses a wizard dialog to query the user to declare the primary changes and mark whether interface changes will propagate. The dialog guides the user stepwise through connected system parts to gather follow-up changes.

Let's briefly compare both derivation approaches. The benefit of using derivation from architecture model changes is that the user, i.e., software architect, can use a simple and familiar architecture model editor. However, there are changes that do not affect the architecture and that can not be derived by changing the architecture model. On the other hand the wizard dialog is something new to the user but can handle activities that are not visible by changing the architecture model. Therefore we recommend a hybrid usage of wizard guidance and architecture modeling.

3.5 Bottom-Up Effort Estimation

The work plan contains the split-up work activities. KAMP uses a bottom-up effort estimation approach for gathering the time effort estimates. In other words, the developers are asked to give time effort estimates for each work activity. KAMP aggregates all effort estimates to a single number at the work plan level.

The benefit of using bottom-up estimation is that a calibration of model parameters from historical data is not necessary since people consider their own productivity implicitly when giving estimates. Nevertheless, the approach is open to be connected to parametric estimation approaches such as Function Point or COCOMO.

4 Initial Case-Study

We implemented the approach as tool in the Eclipse environment and integrated it with the rest of Q-ImPrESS tool chain. In order to show the applicability of the approach we used KAMP on a case study. For this purpose we used the Enterprise SOA showcase, [10], that is one of the demonstration systems of the Q-ImPrESS project.

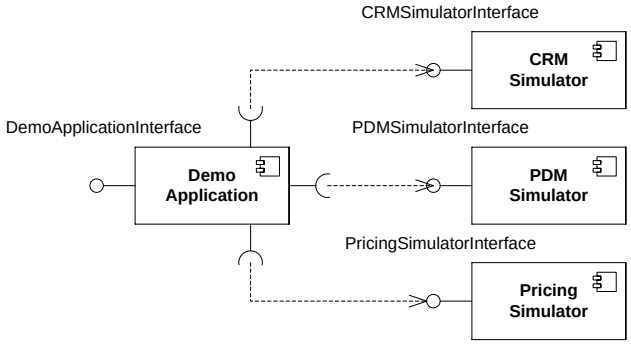


Figure 1. UML component diagram of Enterprise SOA showcase

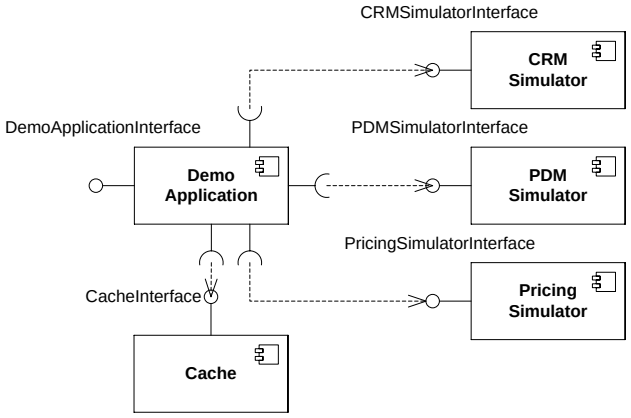


Figure 2. UML component diagram of Enterprise SOA showcase with cache

4.1 System description

The Enterprise SOA showcase consists of several small software systems implementing basic processes in the area of Supply Chain Management and Order Management. Its focus lies on the interaction between those software systems without providing full implementation of the various processes. Also components for simulating the usage of particular software systems are provided. Most of the systems consist of a database, a web front-end and web services for remote access.

The core systems of the Enterprise SOA showcase are CRM (Customer Relationship Management System), PDM (Product Data Management System), Pricing Engine and Inventory System. The simulation systems are Order Simulator, Shipment Simulator and Simulation Manager. Finally there is a Demo application for retrieving information from CRM, PDM and Pricing through Web Services.

A part of the system as UML component diagram is shown in Figure 1.

4.2 Architecture models

Our project partner, Itemis, that is responsible for the showcase, created an architecture model of the system. We refer to this model as the main alternative of the system. For the Enterprise SOA showcase we collected a set of change requests, which are anticipated during system evolution. For illustration purposes in this paper we selected one change request that is described in the following. For this change request a subalternative model has been created in the architecture model evolution hierarchy of the Q-ImPrESS tool chain.

4.3 Change request: Introduce Cache to Demo Application

Change request specification Due to performance issues with respect to the Demo application the following change request arises. A cache should be inserted. The Demo application manager should ask a cache for query results. Only in case of cache miss it should submit requests to the web services of the other subsystems CRM, PDM and Pricing.

Scenario modelling KAMP is utilized to determine a work plan for this change request. Therefore a subalternative architecture model is created and adapted according to the change request.

Here we list the steps done in the model editor: The cache component (Cache) is inserted into the repository. A cache interface (CacheInterface) with three operations (getQueryResult, putQueryResult, clear) for putting and getting of values and clearing the cache is specified in the repository. The cache component gets a provided interface port of type CacheInterface. A subcomponent instance of type Cache is created. A required interface port of type CacheInterface is added to the Demo application manager component. A connector is drawn that links provided and required interface ports of Cache component and Demo application manager component.

Besides the structural changes the architects adapt the dynamics. As a result the provided operation queryPrice of the Demo application manager is modified. The control flow is adapted by inserting a branch action to differentiate the cases of cache hit and cache miss.

An UML component diagram of the changed static structure is presented in Figure 2

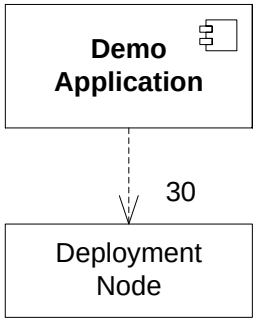


Figure 3. Deployment diagram for DemoApplication component. Component is deployed on 30 nodes.

Work plan derivation After creating the subalternative model that represents the target model after the change request is implemented the architect starts the KAMP derivation process. KAMP calculates a differences model between the mainalternative model files and the subalternative model files. The resulting work plan from the derivation process is shown in Listing 1.

Listing 1. Workplan for Change Request

```
Add InterfaceDefinition CacheInterface
  Add OperationDefinition getQueryResults
  Add OperationDefinition putQueryResults
  Add OperationDefinition clear
Add Component Cache
  Add Provided InterfaceImplementation
       CacheInterface
    Add OperationImplementation
         getQueryResults
    Add OperationImplementation
         putQueryResults
    Add OperationImplementation clear
Modify Component DemoApplication
  Modify Provided InterfaceImplementation
       DemoApplicationInterface
    Modify OperationImplementation queryPrice
```

Deriving deployment activities As can be seen in the work plan the component DemoApplication has to be modified. From the information present in the deployment view of the architecture model (see Figure 3) KAMP retrieves that this component is allocated to 30 nodes. Hence, KAMP adds a new activity to the work plan: Redeploy component DemoApplication (on 30 Nodes). As a result, the modification of a component leads to the follow-up effort for redeployment of the components.

Effort Estimation Our project partners annotated the work plan activities with time effort estimates in Person Days. The aggregated time efforts are then exported to the

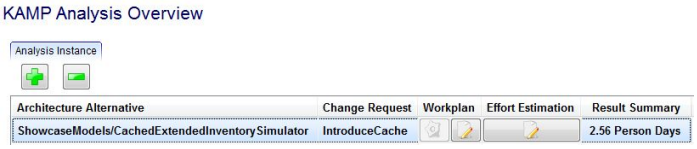


Figure 4. Result overview

Q-ImPrESS result model which can be used as input for trade-off analysis. A screenshot of the result overview is shown in Figure 4. The results are given in Person Days.

5 Related work

5.1 Scenario-Based Architecture Quality Analysis

In literature there are several approaches which analyze quality of software systems based on software architectures. In the following paragraphs we discuss approaches which make explicitly use of scenarios. There are already two survey papers ([1], [9]) which summarize and compare existing architecture evaluation methods.

Software Architecture Analysis Method (SAAM) [7] SAAM was developed in 1994 by Rick Kazman, Len Bass, Mike Webb and Gregory Abowd at the SEI as one of the first methods to evaluate software architectures regarding their changeability (as well as to related quality properties, such as extensibility, portability and reusability). It uses an informally described architecture (mainly the structural view) and starts with gathering change scenarios. Then via different steps, it is tried to find interrelated scenarios, i.e., change scenarios where the intersection of the respective sets of affected components is not empty. The components affected by several interrelated scenarios are considered to be critical and deserve attention. For each change scenario, its costs are estimated. The outcome of SAAM are classified change scenarios and a possibly revised architecture with less critical components.

The Architecture Trade-Off Analysis Method (ATAM) [7] ATAM was developed by a similar group for people from the SEI taking into account the experiences with SAAM. In particular, one wanted to overcome SAAM's limitation of considering only one quality attribute, namely, changeability. Much more, one realised that most quality attributes are in many architectures related, i.e., changing one quality attribute impacts other quality attributes. Therefore, the ATAM tries to identify trade-offs between different quality attributes. It also expands the SAAM by giving more guidance in finding change scenarios. After these are identified, each quality attribute is firstly analysed in isolation. Then, different to SAAM, architectural decisions are identified and the effect (sensitivity) of the design decisions on each quality attribute

is tried to be predicted. By this "sensitivity analysis" one systematically tries to find related quality attributes and trade-offs are made explicit. While the ATAM provides more guidance as SAAM, still tool support is lacking due to informal architectural descriptions and the influence of the personal experience is high. (Therefore, more modern approaches try to lower the personal influence, e.g., POSAAM [8].) Different to our approach, change effort is not measured as costs on ATAM.

The Architecture-Level Prediction of Software Maintenance (ALPSM) [4] ALPSM is a method that solely focuses on predicting software maintainability of a software system based on its architecture. The method starts with the definition of a representative set of change scenarios for the different maintenance categories (e.g. correct faults or adapt to changed environment), which afterwards are weighted according to the likelihood of occurrence during the systems's lifetime. Then for each scenario, the impact of implementing it within the architecture is evaluated based on component size estimations (called scenario scripting). Using this information, the method finally allows to predict the overall maintenance effort by calculating a weighted average of the effort for each change scenario. As a main advantage compared to SAAM and ATAM the authors point out that ALPSM neither requires a final architecture nor involves all stakeholders. Thus, it requires less resources and time and can be used by software architects only to repeatedly evaluate maintainability. However, the method still heavily depends on the expertise of the software architects and provides little guidance through tool support or automation. Moreover, ALPSM only proposes a very coarse approach for quantifying the effort based on simple component size measures like LOC.

The Architecture-Level Modifiability Analysis (ALMA) [5]
The ALMA method represents a scenario-based software architecture analysis technique specialized on modifiability and was created as a combination of the ALPSM approach [4] with [13]. Regarding the required steps, ALMA to a large extend corresponds to the ALPSM approach, but features two major advantages. First, ALMA supports multiple analysis goals for architecture-level modifiability prediction, namely maintenance effort prediction, risk estimation and comparison of architecture alternatives. Second, the effort or risk estimation for single change scenarios is more elaborated as it explicitly considers ripple effects by taking into account the responsible architects' or developers' expert knowledge (bottom up estimation technique). Regarding effort metrics, ALMA principally allows for the definition of arbitrary quantitative or qualitative metrics, but the paper itself mainly focuses on lines of code (LOC) for expressing component size and complexity of modification (LOC/month). Moreover, the approach as presented in the paper so far only focuses on modifications relating to software development activities (like component (re-)implementation), but does not take into account software management activities, such as re-deployment, upgrade installation, etc.

5.2 Change Effort Estimation

Top-Down Effort Estimation Approaches in this section estimate efforts in top-down manor. Although they are intended for forward engineering development projects, one could also assume their potential applicability in evolution projects. Starting from the requirement level, estimates about code size are made. Code size is then related somehow to time effort. There are two prominent representatives of top-down estimation techniques: *Function Point Analysis (FPA)* [11] and *Comprehensive Cost Model (COCOMO) II* [6]. COCOMO-II contains three approaches for cost estimation, one to be used during the requirement stage, one during early architectural design stage and one during late design stage of a project. Only the first one and partially the second one are top-down techniques. Although FPA and COCOMO-II-stage-I differ in detail, their overall approach is sufficiently similar to be treated commonly in this paper. In both approaches, the extent of the functionality of a planned software system is quantified by the abstract unit of function points (called "applications points" in COCOMO). Both approaches provide guidance in counting function points given an informal requirements description. Eventually, the effort is estimated by dividing the total number of function points by the productivity of the development team. (COCOMO-II-stage-I also takes the expected degree of software reuse into account.) In particular COCOMO-II in the later two stages takes additional information about the software development project into account, such as the degree of generated code, stability of requirements, platform complexity, etc. Interestingly, architectural information is used only in a very coarse grained manner (such as number of components). Both approaches require a sufficient amount of historical data for calibration. Nevertheless, it is considered hard to make accurate predictions with top-down estimations techniques. Even Barry Boehm (the author of COCOMO) notes that hitting the right order of magnitude is possible, but no higher accuracy[1].

Bottom-Up Effort Estimation – Architecture-Centric Project Management [14]
(ACPM) is a comprehensive approach for software project management which uses the software architecture description as the central document for various planning and management activities. For our context, the architecture based cost estimation is of particular interest. Here, the architecture is used to decompose planned software changes into

[1] http://cost.jsc.nasa.gov/COCOMO.html

several tasks to realise this change. This decomposition into tasks is architecture specific. For each task the assigned developer is asked to estimate the effort of doing the change. This estimation is guided by pre-defined forms. Also, there is no scientific empirical validation. But one can argue that this estimation technique is likely to yield more accurate prediction as the aforementioned top-down techniques, as (a) architectural information is used and (b) by asking the developer being concerned with the execution of the task, personal productivity factors are implicitly taken into account. This approach is similar to KAMP by using a bottom-up estimation technique and by using the architecture to decompose change scenarios into smaller tasks. However, KAMP goes beyond ACPM by using a formalized input (architectural models must be an instance of a predefined meta-model). This enables tool-support. In addition, ACPM uses only the structural view of an architecture and thus does not take software management costs, such as re-deployment into account.

6 Conclusions

In this paper we presented the KAMP approach for estimating the evolution effort of a given change request based on the architectural model of a service-oriented software system. The main contributions of our method are:

- a way to map change requests to architecture models and derive a work plan by calculating differences between models, enhanced with user inputs from a wizard dialog and
- an integrated bottom-up estimation approach providing evolution effort estimations, which are not limited to implementation efforts only.

We showed the applicability of our approach by using it on the Enterprise SOA Showcase, an open-source industrial demonstration systems developed within the Q-ImPrESS project.

7 Acknowledgements

The work presented in this paper was funded within the Q-ImPrESS research project (FP7-215013) by the European Union under the Information and Communication Technologies priority of FP7.

References

[1] M. Babar, L. Zhu, and R. Jeffery. A framework for classifying and comparing software architecture evaluation methods. *Software Engineering Conference, 2004. Proceedings. 2004 Australian*, pages 309–318, 2004.

[2] S. Becker, L. Bulej, T. Bures, P. Hnetynka, L. Kapova, J. Kofron, H. Koziolek, J. Kraft, R. Mirandola, J. Stammel, G. Tamburelli, and M. Trifu. Q-ImPrESS Project Deliverable D2.1: Service Architecture Meta Model (SAMM). Technical Report 1.0, Q-ImPrESS consortium, September 2008. http://www.q-impress.eu/wordpress/wp-content/uploads/2009/05/d21-service_architecture_meta-model.pdf.

[3] S. Becker, M. Trifu, and R. Reussner. Towards Supporting Evolution of Service Oriented Architectures through Quality Impact Prediction. In *1st International Workshop on Automated engineeRing of Autonomous and run-time evolving Systems (ARAMIS 2008)*, September 2008.

[4] P. Bengtsson and J. Bosch. Architecture level prediction of software maintenance. *Software Maintenance and Reengineering, 1999. Proc. of the Third European Conference on*, pages 139–147, 1999.

[5] P. Bengtsson, N. Lassing, J. Bosch, and H. van Vliet. Architecture-level modifiability analysis (alma). *Journ. of Systems and Software*, 69(1-2):129 – 147, 2004.

[6] B. W. Boehm, editor. *Software cost estimation with Cocomo II*. Prentice Hall, Upper Saddle River, NJ, 2000.

[7] P. Clements, R. Kazman, and M. Klein. *Evaluating software architectures*. Addison-Wesley, 4. print. edition, 2005.

[8] D. B. da Cruz and B. Penzenstadler. Designing, Documenting, and Evaluating Software Architecture. Technical Report TUM-INFO-06-I0818-0/1.-FI, Technische Universität München, Institut für Informatik, jun 2008.

[9] E. Dobrica, L.; Niemela. A survey on software architecture analysis methods. *Transactions on Software Engineering*, 28(7):638–653, Jul 2002.

[10] C. Häcker, A. Baier, W. Safonov, J. Tysiak, and W. Frank. Q-ImPrESS Project Deliverable D8.6 Enterprise SOA Showcase initial version. Technical Report 1.0, Q-ImPrESS consortium, January 2009.

[11] IFPUG. *Function Point Counting Practices Manual*. International Function Points Users Group: Mequon WI, 1999.

[12] ISO/IEC. Software Engineering - Product Quality - Part 1: Quality. *ISO/IEC 9126-1:2001(E)*, Dec 1990.

[13] N. Lassing, D. Rijsenbrij, and H. van Vliet. Towards a broader view on software architecture analysis of flexibility. *Software Engineering Conference, 1999. (APSEC '99) Proceedings. Sixth Asia Pacific*, pages 238–245, 1999.

[14] D. J. Paulish and L. Bass. *Architecture-Centric Software Project Management: A Practical Guide*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

Preparing for a Literature Survey of Software Architecture using Formal Concept Analysis

Luís Couto and José Nuno Oliveira
CCTC
Departamento de Informática
Universidade do Minho
Braga, Portugal
{*pg15260@alunos.uminho.pt, jno@di.uminho.pt*}

Miguel Ferreira and Eric Bouwers
Software Improvement Group
Amsterdam, Netherlands
{*m.ferreira, e.bouwers*}*@sig.eu*

***Abstract*—The scientific literature on Software Architecture (SA) is extensive and dense. With no preparation, surveying this literature can be a daunting task for novices in the field. This paper resorts to the technique of Formal Concept Analysis (FCA) in organizing and structuring such a body of knowledge.**

We start by surveying a set of 38 papers bearing in mind the following questions: "*What are the most supported definitions of software architecture?*", "*What are the most popular research topics in software architecture?*", "*What are the most relevant quality attributes of a software architecture?*" and "*What are the topics that researchers point out as being more interesting to explore in the future?*". To answer these questions we classify each paper with appropriate keywords and apply FCA to such a classification. FCA allows us to structure our survey in the form of *lattices* of concepts which give evidence of main relationships involved.

We believe our results will help in guiding a more comprehensive, in-depth study of the field, to be carried out in the future.

***Keywords*-Introductory and Survey; Software Architectures; Formal Concept Analysis;**

I. Introduction

Architecture is a relevant aspect of software systems because it "*allow[s] or preclude[s] nearly all of the systems quality attributes*" [38]. Although the term "architecture", referring to software, seems to be widely accepted in both academia and industry we notice that there is no consensual understanding of it. A quick overview of just a few papers will reveal significantly different definitions of SA, for instance. This calls for knowledge classification and aggregation in the field. Our main motivation in this paper is to use formal concept analysis (FCA) [39] to help us in classifying and structuring the vast bibliography in the area, paving the way to the future construction of a taxonomical body of knowledge. We consider that a survey could help to clear this and other issues, such as what the main topics of the field are, what quality attributes are more relevant to consider when working with the architecture of a software system, or what the most promising topics for future research are. Before starting a full fledge literature review of the field (as proposed in [40] for instance), we propose to do a preliminary study to shed some light on these, and possibly other, research questions (RQs). With such a preliminary review and analysis one can get a better focus on subsequent reviews due to a better understanding of how the field is partitioned in streams of research, who is working on what and how different topics relate to each other.

Our main assumption is that it is possible to use FCA to answer questions. FCA is a mathematical theory for data analysis that derives ontologies from sets of objects and their properties. It can be used to explore a domain, reason about it or simply describe it in a structured way. We rely on FCA visualization capabilities to help reason about the field to be surveyed.

The contributions of this paper are:

- the exemplification of how can FCA be used in preparing research literature reviews;
- answers to the RQs used to illustrate the technique.

For the illustration of the preliminary survey approach, we formulate the following RQs about SA.

1) What are the most supported definitions of software architecture?
2) What are the most popular research topics in software architecture?
3) What are the most relevant quality attributes of a software architecture?
4) What are the topics that researchers point out as being more interesting to explore in the future?

The remainder of this paper is structured as follows. Section II introduces FCA and the lattices used later on in the paper. Section III describes the proposed approach to a preliminary literature review. Each of the following four sections is an illustration of how the generic approach can be applied to a specific RQ. The paper terminates with a discussion of the outcome of the study in Section VIII.

II. Formal Concept Analysis

This section provides background on FCA in general, and the interpretation of the lattices used in the paper. Readers familiar with FCA may want to skip this section altogether.

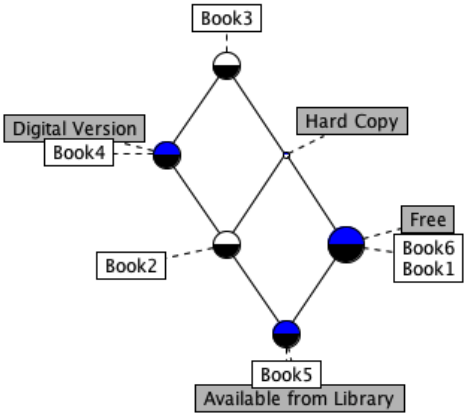


Figure 1. Example concept lattice

FCA comes from the field of applied mathematics and it is described by its proponent as follows:

> *The aim and meaning of Formal Concept Analysis as mathematical theory of concepts and concept hierarchies is to support the rational communication of humans by mathematically developing appropriate conceptual structures which can be logically activated.* [39]

The input for FCA is a *context*, which is a relation (typically a matrix) between *objects* and *attributes*. This relation forms an ontology of *concepts* and their *relationships*. A concept can be considered a *subconcept* if its *extension* (the set of its objects) is contained in the extension of its *superconcept* or, dually, if its *intension* (the set of its attributes) contains the intention of its superconcept [39].

Although FCA provides advanced features for knowledge exploration, in our analysis we only use it for visualizing *concept lattices*, which provide a structured view of the concepts under study. We are interested in understanding which objects (papers in our case) relate to which attributes (the different attributes we defined for each RQ), what clusters of objects and attributes are there, and what hierarchical relations exist among concepts. We refer the reader to a paper [41] where FCA was used for the same purpose.

To create the concept lattices in this study, we chose to use the "The Concept Explorer" (`conexp`) [1] tool. Let us start by showing how to interpret a concept lattice in FCA. Figure 1 depicts one such concept lattice where books (objects) are related to their properties (attributes). The figure shows 6 concepts (nodes in the lattice) of 3 different types. White labels represent objects, whereas gray labels represent attributes. The dashed lines between concepts and labels are a visual aid to help identify to which concepts labels belong. Solid lines represent the super/sub-concept relationship. Concepts depicted by half white, half black circles only refer to objects, meaning that objects in such nodes have empty intentions (such is the case of the top most concept where *Book3* sits) because it is not related to any of the attributes, or that the objects in that concept inherit the attributes from their superconcepts (the case of the concept where *Book2* sits, which inherits the attributes DIGITAL COPY and HARD COPY from its two superconcepts). Concepts depicted by half blue, half black circles refer to both objects and attributes (see e.g. the concept where *Book4* sits). The objects in these concepts are related to the attributes that sit on the same concept, and to the attributes of their superconcepts. Finally, the concepts depicted by a rather small and empty circle can refer to attributes (the case of the concept where the attribute HARD COPY sits), or merely serve as hierarchical links between their superconcepts and their subconcepts.

[1] http://conexp.sourceforge.net

III. GENERIC APPROACH

The approach proposed in this paper is carried out in four steps: paper selection, paper classification, use of FCA, and analysis of results. This section gives an overview of these steps.

A proper selection of papers should have strictly defined criteria for searching and filtering papers. However, because this is simply a preliminary study we skipped such a structured selection and from a set of 4 papers we chased bibliography references until we reached a larger set that both contained different types of papers (such as surveys, evaluations, new proposals, etc), and covered several different topics within the field of SA. We ended up with a selection of 38 papers published between 1992 and 2010. This set of papers is not meant to be fully representative of the field of SA, but it is vast enough for a preliminary study such as this one.

With the set of papers to survey established it is necessary to classify each paper according to the attributes selected for each of the RQs. Different sets of attributes are used to answer different RQs. For each RQ, a first (more extensive) set of attributes is manually collected from all papers. The collected attributes were found in abstract, keywords, introduction, future work and conclusion sections. Then, the related attributes are grouped together in order to reduce the number of attributes in the final set. Attributes are dropped if too few papers support them. This is typically the case for attributes that are not related to more than 1 or 2 papers whenever there are more attributes that get related to a significantly larger number of papers. Once this set is stable, then a relation is built between each paper and the attributes that are relevant to that paper.

The following step is to apply FCA. There are several tools that implement FCA. As mentioned previously, we use `conexp`, an open source FCA tool.

Finally, the resulting concept lattices are analyzed and interpreted, so as to focus on the structural relations they

uncover between papers and attributes.

The following four sections report the results of applying this generic approach to the selected RQs. The set of papers is the same for all the RQs, except for one where it was further reduced.

IV. What are the most supported definitions of software architecture?

A. Attributes

The attributes for this RQ are the following.

Design	SA is the set of design decisions, specification or otherwise abstract representation of the software system.
Structure	SA is the structure of components, their relationship and external properties.
Constraints	SA defines the constraints on the realization or satisfaction of properties of its elements, or on the legal and illegal patterns in the communication between elements.
Quality	SA influences and/or determines quality attributes of software.

B. Results

The lattice of Figure 2 shows that a group of 8 papers does not support any of the 4 given definitions of SA, as they sit in the topmost concept. Some papers are related to just one definition of SA. These are the papers that inhabit the concepts that also hold the attributes QUALITY (3 papers), DESIGN (6 papers) and STRUCTURE (11 papers). All other papers support two definitions simultaneously. Regarding attributes, the odd one out is CONSTRAINTS because no paper supports it alone. The papers that support the definition CONSTRAINTS also support DESIGN or STRUCTURE. Finally, no paper supports more than two definitions.

C. Discussion

The initial scan of the papers revealed 15 definitions. This number was, however, reduced to 4. Some definitions were simply dropped because too few papers supported them, and others were merged together. An example of such a merge is the case of STRUCTURE, which aggregates three of the initial definitions, namely STRUCTURE, DECOMPOSITION and RELATIONSHIP. The reason for this aggregation is that most papers that support these definitions articulate them together in sentences like "*...software architecture of a program or computer system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them*" *Bengtson et al. 2004* [8].

We found that 8 papers do not support any of the analyzed definitions, 5 of these not providing any definition at all. The remaining 3 papers do provide definitions, but these were dropped. Eight of the papers that provide a definition for SA quote it from elsewhere.

It turns out that STRUCTURE is the most supported (17 papers) definition among the papers we surveyed, followed by DESIGN (13 papers). Recall the definitions of DESIGN and STRUCTURE from Section IV-A. These definitions, although not exactly the same, look fairly similar to us. DESIGN refers to decisions and abstract representations. On the other hand, STRUCTURE refers to a structure of components that are related to each other in some way. We think that the decomposition of a software system in different components that are structurally related can be viewed as an abstract representation of that system. Also, such structure of components exists due to decisions made by people developing the system. This understanding of the definitions, however, does not make them the same thing. The structure of components can be observed at different abstraction levels, such as the structure of source code packages (or equivalent) which are already highly concrete and far away from the abstract representation. The point is that SA viewed from the STRUCTURE perspective can be too detailed to be considered abstract. All in all, we believe that although there might be some overlapping these are two different definitions of SA.

V. What are the most popular research topics in software architecture?

A. Attributes

The attributes for this RQ are the following.

Notation	Addresses a notation (or description language) for representing SA.
Method	Addresses a method for analysis or evaluation of SAs.
Tool	Addresses a tool for the analysis or evaluation of SAs.
Evaluation	Pertains to SA evaluation with respect to one or more quality attributes.
Analysis	Pertains to SA analysis not leading to appreciation of quality attributes.
Scenarios	Addresses scenarios as a technique for evaluation or analysis of SA.
Metrics	Addresses metrics as tools for evaluation or analysis of SA.
Reviews	Addresses reviews as tools for evaluation or analysis of SA.
Prototypes	Addresses prototypes as tools for either evaluation or analysis of SA.

B. Results

Due to the overly complex concept lattice obtained if using the entire attribute set we decided for simplification in detriment of a complete view of the concepts. To this end, we consider a core set of topics containing NOTATION, EVALUATION, ANALYSIS, TOOL and METHOD. This leaves out SCENARIOS, METRICS, REVIEWS and PROTOTYPES as

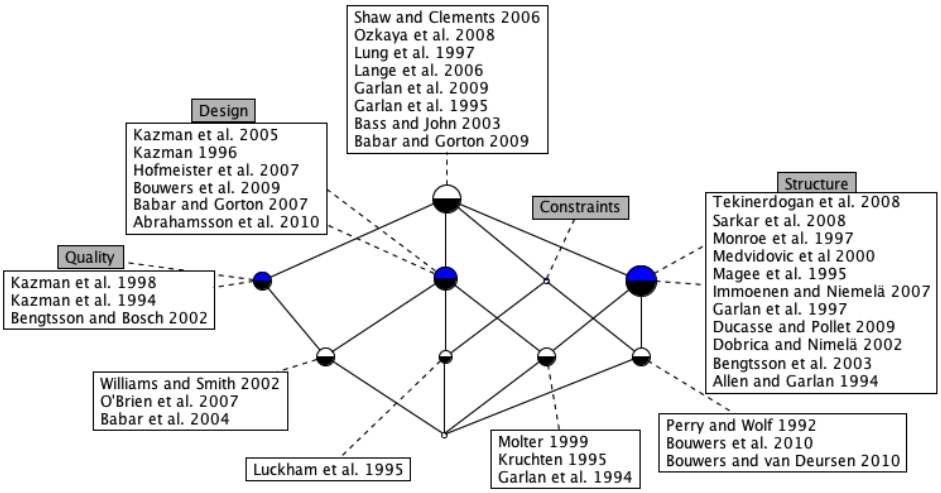


Figure 2. Concept lattice for software architecture definitions.

we expect these to be bound to some of the topics of the core set.

Figure 3 depicts the concept lattice built from classifying each surveyed paper according to the attributes of the core set. This lattice shows that there is one paper (*Shaw and Clements 2006* [35]) that does not cover any of the 6 analyzed topics. In the lattice it is visible that 5 concepts descend directly from the topmost. For easy reference we will refer to these as *level1* concepts. Four of these hold both objects and attributes, and one only holds the attribute TOOL, which means that no paper covers the TOOL topic alone. On the other hand, the 4 that hold both objects and attributes reveal that some papers are focused on just one topic. These topics are EVALUATION (9 papers), NOTATION (6 papers), METHOD (1 paper) and ANALYSIS (4 papers). Moving further down along the lattice it becomes clear that the number of connections among concepts increases. Two of the concepts that descend directly from *level1* concepts hold papers, and both have two direct superconcepts. *Kazman 1996* [20] covers TOOL and ANALYSIS, whereas a group of 10 papers cover EVALUATION and METHOD. Analyzing the lattice from the bottom up, there are 4 concepts that are superconcepts of the bottommost concept and only hold papers. All these papers cover some combination of three topics. No paper covers all topics and every topic is associated to at least one paper.

Adding METRICS to the analysis produces the concept lattice depicted in Figure 4 in which we see that METRICS are covered by 4 papers. All of these papers cover METRICS in addition to some other topic(s). For example *Bouwers et al. 2009* [10] also covers EVALUATION.

Taking SCENARIOS into account together with the core set yields the lattice of Figure 5. Nine papers cover SCENARIOS. A group of 6 papers cover it together with EVALUATION and METHOD. *Lung et al. 1997* [26] is not part of this set because it also covers NOTATION. *Kruchten 1995* [23] covers SCENARIOS together with ANALYSIS, METHOD and NOTATION. Finally, *Kazman 1996* [20] also covers TOOL and ANALYSIS.

The lattice resulting from adding REVIEWS to the analysis is depicted in Figure 6. REVIEWS are covered only by 3 papers, all also covering EVALUATION and METHOD.

PROTOTYPES, the last attribute, generates the lattice of Figure 7. Only *Luckham et al. 1995* [25] covers this attribute and it does so in addition to NOTATION.

C. Discussion

EVALUATION is the topic gathering most papers (21), followed by METHOD (17) and NOTATION (11). The difference in the number of papers that cover EVALUATION (21) and the number of papers that cover ANALYSIS (8) seems to indicate that the research community represented in the surveyed papers believes that attributing quality notions to SA is of paramount importance. On the other hand, it could also mean that there are sufficiently mature analysis techniques available and that quality attributes are the next logical step. The most frequent (12 papers) combination of two topics is EVALUATION and METHOD, which indicates that a good deal of papers focuses on methods for evaluation of SAs. Except for *Kazman 1996* [20] all papers that cover more than one topic, cover METHOD. This indicates that methods for SA evaluation or analysis are still a hot topic in the field. It could be seen as a consequence of the definition of the attribute METHOD. However the attribute TOOL was defined in a similar way and does not gather as many papers. This is perhaps a sign of method immaturity (too early for tool development).

With the exception of one paper (*Kazman 1996* [20]), all other papers that cover METRICS also cover EVALUATION. This reveals that researchers consider metrics to be adequate

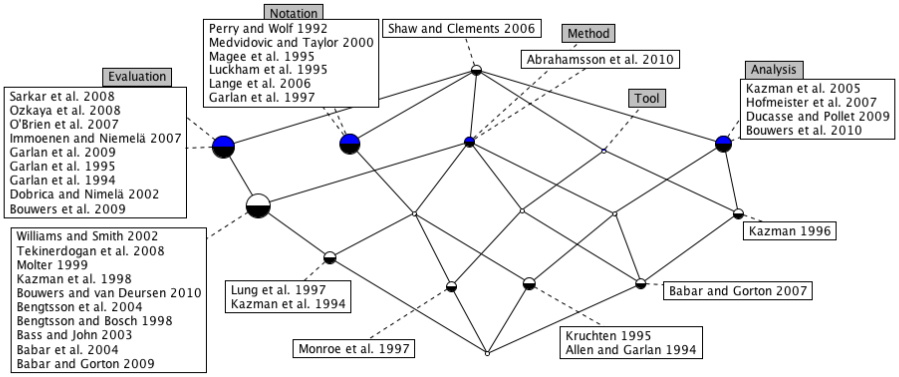


Figure 3. Concept lattice for the core set of research topics.

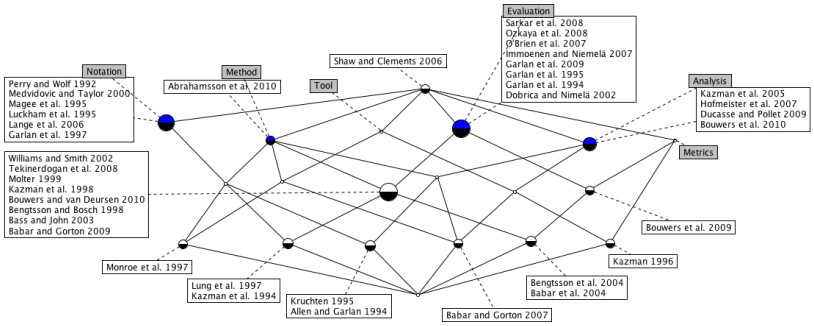


Figure 4. Concept lattice for the core set of research topics plus METRICS.

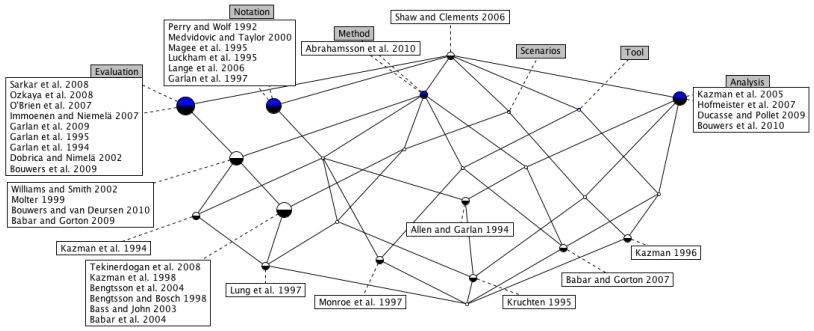


Figure 5. Concept lattice for the core set of research topics plus SCENARIOS.

indicators (or predictors) for quality attributes. Similarly to METRICS, REVIEWS are only covered together with EVALUATION and METHOD. When compared to METRICS and REVIEWS, SCENARIOS are more diversified in the field. Figure 5 shows that SCENARIOS are related to all other topics in the core set, which implies a broad scope of applications. Finally, PROTOTYPES are always covered together with NOTATION and nothing else. This does not mean researchers find prototyping inadequate for evaluation or analysis. Instead, modeling and building prototypes seems

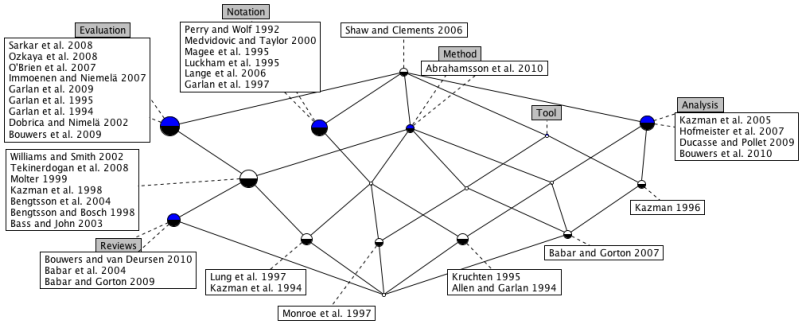


Figure 6. Concept lattice for the core set of research topics plus REVIEWS.

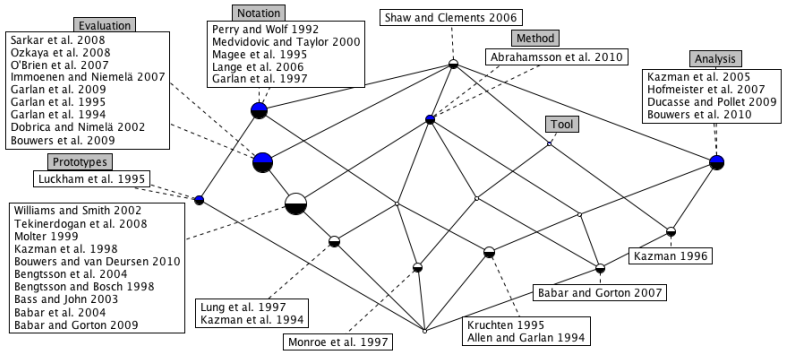
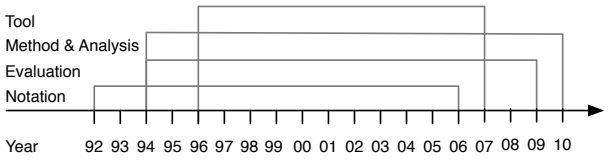


Figure 7. Concept lattice for the core set of research topics plus PROTOTYPES.

in most cases inherent to any of these activities. This could explain why researchers do not mention it explicitly when covering topics such as EVALUATION, ANALYSIS, METHOD or TOOL.



Timeline of researched topics.

From the surveyed papers we do not see a clear evolution in research topics, meaning that most topics were and are researched in parallel. As depicted in the above timeline all topics overlap along most of the timeline.

VI. WHAT ARE THE MOST RELEVANT QUALITY ATTRIBUTES OF A SOFTWARE ARCHITECTURE?

A. Attributes

The attributes for this RQ are the following.

Maintainability How easily can software be maintained.

Usability How easily can software be used.

Reusability How easily can software (or parts of it) be re-used.

Performance The amount of useful work accomplished by software compared to the time and resources used.

Reliability The capability of software to maintain performance under stated conditions for a stated period of time.

Availability The capability of a software product to be in a functioning state.

Security The capability of software to prevent unintended usage and unintended access to its data.

Modifiability How easily can software be modified.

Interoperability How easily can software interact with other systems.

Testability How easily can software be tested.

Scalability The capability of software to handle growing work loads without prejudice of its service quality.

Generic	The quality attribute is a parameter in the evaluation.

B. Results

For this RQ the set of papers was reduced to the 22 papers that cover the topic EVALUATION which, by definition of the topic, refer to some quality attribute, leading to the lattice of Figure 8. This lattice shows that all papers cover at least one quality attribute. Out of the 22 papers 18 cover at most one quality attribute (including the GENERIC). Which only leaves 4 papers that cover more than one attribute. Two papers, *Immonen and Niemelä* [19] and *Lung et al. 1997* [26], cover two quality attributes (respectively RELIABILITY and AVAILABILITY, and MODIFIABILITY and REUSABILITY). *Babar et al. 2004* [5] covers 6 quality attributes. Finally, the paper *O'Brien et al. 2007* [31] covers 9 attributes. From the perspective of the quality attributes, three (SCALABILITY, TESTABILITY, and INTEROPERABILITY) are covered by a single paper (*O'Brien et al. 2007* [31]). SECURITY is always covered together with other quality attributes (RELIABILITY, AVAILABILITY, PERFORMANCE, and USABILITY). AVAILABILITY is always covered together with RELIABILITY. The last odd attribute is GENERIC for which the associated papers do not cover any other quality attribute (this is actually a meta quality attribute as explained in Section VI-A).

C. Discussion

The fact that AVAILABILITY and RELIABILITY are always covered together hints at the similarities between these two quality attributes. In fact a software system that is very reliable should have no problems in being available to its users. However, the contrary might not be true, since a system can be available but in the end not so reliable. For instance a system could be available for its users to perform whatever tasks they need it for, but still lose important data without the user perceiving it. This seems to indicate AVAILABILITY as an aspect of RELIABILITY.

Because SECURITY is always covered together with RELIABILITY, AVAILABILITY, PERFORMANCE and USABILITY it does not seem a main concern of researchers focusing on SA. The fact that it is covered by no more than two papers seems to reinforce this interpretation.

Seven papers cover GENERIC evaluations, meaning that the quality attribute is a parameter of the evaluation method, therefore these methods can be applied to different quality attributes. The next most covered quality attribute is REUSABILITY (6 papers) which indicates this quality attribute as the most relevant for the surveyed papers. This makes sense if one considers that most SAs are built from scratch every time, over and over. Just like with code libraries, researchers believe that there should be ways to reutilize existing SAs, or at least bits and pieces. Lastly, we have MAINTAINABILITY and RELIABILITY each with 4.

The remaining quality attributes are covered by three or two papers, with the exception of INTEROPERABILITY and TESTABILITY. A possible explanation for these two quality attributes to appear in isolation from the others, could be the development stage of the SA they apply to. There is a distinction between *designed* and *implemented* architecture. The former refers to the architecture that was initially designed for a system typically using an Architecture Description Language (ADL), whereas the latter refers to the architecture that got implemented in source code. It could be that both INTEROPERABILITY and TESTABILITY can be better assessed when considering implemented architectures.

VII. WHAT ARE THE TOPICS THAT RESEARCHERS POINT OUT AS BEING MORE INTERESTING TO EXPLORE IN THE FUTURE?

A. Attributes

The attributes for this RQ are the following.

Notation	Improve notation support for SA, including enrichment of semantics.
Knowledge	Create and/or extend a reusable SA body of knowledge.
Validation	Validate usefulness of methods, tools and languages used for evaluation and analysis of SAs.
Evaluation	Improve SA evaluation methods.
Tooling	Improve SA tool support.
Quality	Study SA quality attributes.
Integration	Promote integration of SA evaluation and analysis methods in software development processes.
Measurement	Propose new, or select from existing, metrics for SA.

B. Results

The lattice of Figure 9 shows that all topics sit in concepts that directly descend from the topmost concept. This means that the topics are fairly independent. A group of 6 papers sits in the topmost concept, meaning that they either do not propose any future work, or their topics cannot be found in the attribute set. No paper proposes EVALUATION or TOOLING alone. All other topics are singled out in some papers. Before moving on to the remaining 20 papers that propose several topics, one noteworthy observation is that VALIDATION is the topic which is most proposed together with other topics (11 papers), followed by INTEGRATION (10 papers), and TOOLING (7 papers). The 17 papers that propose multiple topics do so in pairs.

C. Discussion

When compiling the attribute set for this RQ, we encountered two papers (*Perry and Wolf 1992* [33] and *Babar et al. 2004* [5]) which proposed a working definition of

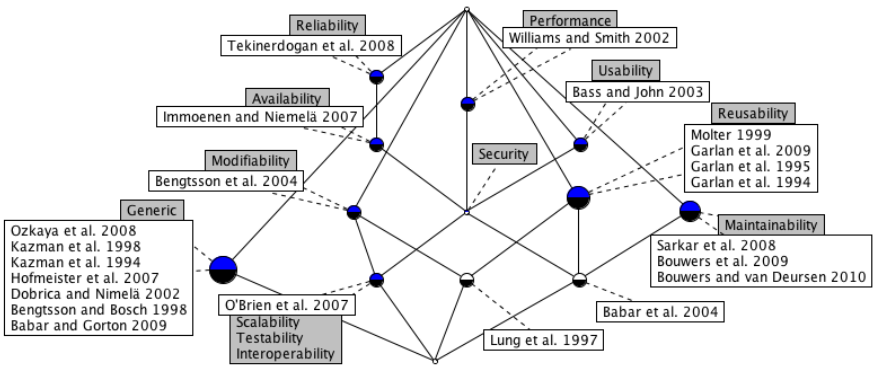


Figure 8. Concept lattice for the most explored quality attributes.

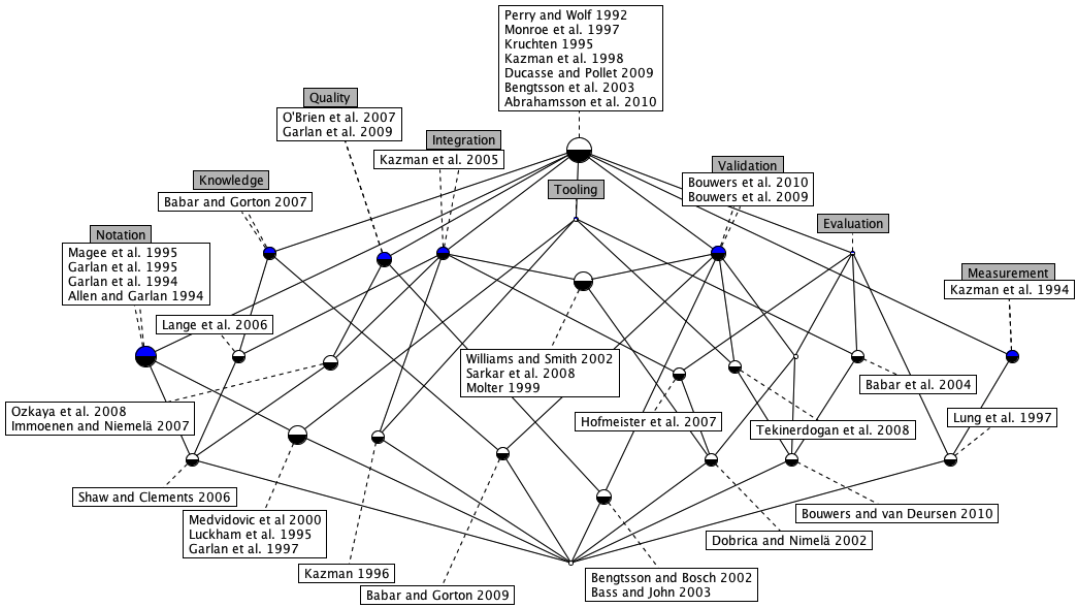


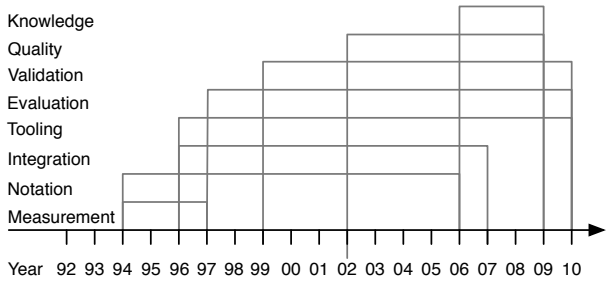
Figure 9. Concept lattice for the research topics that researchers point out as more interesting to explore in the future.

SA. Although this topic did not make it into the analyzed attribute set, it shows the relevance of our first RQ.

The pairs of topics that were pointed out by more papers (3 each) were NOTATION and TOOLING, and INTEGRATION and VALIDATION. This seems to stress the need for tool support for SA specific notations, and that methods and tools need to be both validated with respect to their usefulness and better integrated in development processes.

Six papers do not propose future work topics at all. The most proposed topics VALIDATION and INTEGRATION (11 papers each), by definition, only make sense when researched together with something else. Even though some papers single these two out, we believe that the intention of the authors is to validate or integrate whatever methods or tools they have developed. All in all, it appears that there are many avenues to be explored by new research and that there is no topic that gathers the majority of the researchers.

Contrary to what we found for RQ2, there is not as much time overlapping between proposals for future research topics. The following timeline shows that both MEASUREMENT and KNOWLEDGE span over short periods of time 1994–1997 and 2006–2009, respectively. For the remaining topics there is significant overlap for most of the analyzed period, which means that for this RQ there is no crystal clear evolution in what researchers consider to be important to research in the future.



Timeline of future research topics.

VIII. Conclusions

Our preliminary survey shows that the term SA lacks a clear definition. Two of the surveyed papers actually mention the desirable clarification of the term as future work. From the analysis of RQ1 ("*What are the most supported definitions of software architecture?*") we were able to single out 2 (Design and Structure) of the 4 main definitions as being the most supported. This dichotomy can be explained by the gap existing between the designed architecture (what is initially designed for a system) and the implemented architecture (what is actually implemented in the source code). In a full fledge survey of the field it might be desirable to take both definitions into account, or to just focus on one. If one already knows the field to be surveyed, such differences might be obvious. However, if the field is unknown, a preliminary study as the one described in this paper can help to focus on what to include and what to exclude from the survey.

With respect to RQ2 ("*What are the most popular research topics in software architecture?*"), one thing that stands out is the fact that whenever a paper covers two topics, Method is almost always (there is one exception) one of them. So if the focus of the survey would be "methods for SA", one would already expect not to exclude any topic from the literature selection. On the other hand, should the focus of the survey be "tools for dealing with SA", one would already expect to find more papers regarding methods for architectural analysis that than, for instance, architectural evaluation. Of course, these are not rules of thumb, but they help set the expectations, which in turn guide literature searches and help validate findings. RQ2 also shows how to partition the analysis if the lattice is overly complex.

Regarding RQ3 ("*What are the most relevant quality attributes of a software architecture?*"), we observed that the maintainability, reusability, usability, performance, reliability and modifiability quality attributes seem to be more impacted by decisions taken at the level of SA. Furthermore, other quality attributes, such as scalability, testability and interoperability, seem to be less relevant according to the number of papers that cover those.

Finally, the analysis of RQ4 ("*What are the topics that researchers point out as being more interesting to explore in the future?*") shows that there are many avenues for future work and that there are many proposals for combining these avenues into streams of research. From all the identified future research topics the ones that gather more consensus are validation and integration efforts. This seems to indicate that the field has matured to the point where there is enough confidence in what has been developed thus fur to start embracing integration of such methods and tools into mainstream software engineering practices. On the other hand, the field has not yet matured enough to have provided full confidence in its methods and tools, thus requiring additional empirical validation studies.

We recognize limitations in this preliminary study for a survey, namely the bias introduced in the selection of papers and attributes. To overcome this bias, one should rely on strictly defined criteria for searching and selecting papers, and the elicitation of the attributes. Since the goal of this study is the demonstration of how FCA can help in preparing for a full fledge survey, we do not consider this bias to be a problem. The way to structure such criteria and protocols for searching, classifying and extracting knowledge from scientific literature has already been addressed in systematic literature reviews for software engineering [40].

From this study we conclude that FCA can help in gathering knowledge about a multifaceted field, and better focus a survey. In addition, we believe that this is also true if the target of the survey is something more specific, such as "methods for SA evaluation that use metrics". Validating this hypothesis is yet to be done.

References

[1] P. Abrahamsson, M. Babar, and P. Kruchten, "Agility and architecture: Can they coexist?" *IEEE Softw.*, vol. 27, no. 2, pp. 16 –22, Mar/Apr 2010.

[2] R. Allen and D. Garlan, "Formalizing architectural connection," in *ICSE*, May 1994, pp. 71 –80.

[3] M. Babar and I. Gorton, "A tool for managing software architecture knowledge," in *SHARK/ADI*. IEEE, 2007, p. 11.

[4] M. A. Babar and I. Gorton, "Software architecture review: The state of practice," *IEEE Comp.*, vol. 42, no. 7, pp. 26–32, 2009.

[5] M. A. Babar, L. Zhu, and R. Jeffery, "A framework for classifying and comparing software architecture evaluation methods," in *ASEC*, ser. ASWEC '04. IEEE CS, 2004, pp. 309–.

[6] L. Bass and B. E. John, "Linking usability to software architecture patterns through general scenarios," *JSS*, vol. 66, no. 3, pp. 187 – 197, 2003.

[7] P. Bengtsson and J. Bosch, "Scenario-based software architecture reengineering," in *ICSR*. IEEE, 2002, pp. 308–317.

[8] P. Bengtsson, N. Lassing, J. Bosch, and H. van Vliet, "Architecture-level modifiability analysis (ALMA)," *JSS*, vol. 69, no. 1-2, pp. 129 – 147, 2004.

[9] E. Bouwers and A. van Deursen, "A lightweight sanity check for implemented architectures," *IEEE Softw.*, vol. 27, no. 4, pp. 44 –50, Jul 2010.

[10] E. Bouwers, J. Visser, and A. van Deursen, "Criteria for the evaluation of implemented architectures," in *ICSM*. IEEE, 2009, pp. 73–82.

[11] E. Bouwers, J. Visser, C. Lilienthal, and A. van Deursen, "A cognitive model for software architecture complexity," in *ICPC*. IEEE Comp. Soc., 2010, pp. 152–155.

[12] L. Dobrica and E. Niemelä, "A survey on software architecture analysis methods," *IEEE TSE*, vol. 28, pp. 638–653, Jul 2002.

[13] S. Ducasse and D. Pollet, "Software architecture reconstruction: A process-oriented taxonomy," *IEEE TSE*, vol. 35, no. 4, pp. 573–591, 2009.

[14] D. Garlan, R. Allen, and J. Ockerbloom, "Architectural mismatch: Why reuse is so hard," *IEEE Softw.*, vol. 12, pp. 17–26, Nov 1995.

[15] ——, "Architectural mismatch or why it's hard to build systems out of existing parts," in *ICSE*, 1995, pp. 179–185.

[16] D. Garlan, R. Monroe, and D. Wile, "Acme: an architecture description interchange language," in *CASCON*, ser. CASCON '97. IBM Press, 1997, pp. 7–.

[17] D. Garlan, R. Allen, and J. Ockerbloom, "Architectural mismatch: Why reuse is still so hard," *IEEE Softw.*, vol. 26, no. 4, pp. 66 –69, Jul/Aug 2009.

[18] C. Hofmeister, P. Kruchten, R. L. Nord, H. Obbink, A. Ran, and P. America, "A general model of software architecture design derived from five industrial approaches," *JSS*, vol. 80, no. 1, pp. 106 – 126, 2007.

[19] A. Immonen and E. Niemelä, "Survey of reliability and availability prediction methods from the viewpoint of software architecture," *SSM*, vol. 7, pp. 49–65, 2008.

[20] R. Kazman, "Tool support for architecture analysis and design," in *ISAW-2 and Viewpoints '96 on SIGSOFT '96 Workshops*, ser. ISAW '96. ACM, 1996, pp. 94–97.

[21] R. Kazman, L. Bass, M. Webb, and G. Abowd, "SAAM: A method for analyzing the properties of software architectures," in *ICSE*. IEEE CS, 1994, pp. 81–90.

[22] R. Kazman, L. Bass, M. Klein, T. Lattanze, and L. Northrop, "A basis for analyzing software architecture analysis methods," *SQJ*, vol. 13, pp. 329–355, 2005.

[23] P. Kruchten, "The 4+1 view model of architecture," *IEEE Softw.*, vol. 12, no. 6, pp. 42 –50, Nov 1995.

[24] C. Lange, M. Chaudron, and J. Muskens, "In practice: UML software architecture and design description," *IEEE Softw.*, vol. 23, no. 2, pp. 40 – 46, Mar/Apr 2006.

[25] D. Luckham, J. Kenney, L. Augustin, J. Vera, D. Bryan, and W. Mann, "Specification and analysis of system architecture using rapide," *IEEE TSE*, vol. 21, no. 4, pp. 336 –354, Apr 1995.

[26] C.-H. Lung, S. Bot, K. Kalaichelvan, and R. Kazman, "An approach to software architecture analysis for evolution and reusability," in *CCASCR*, ser. CASCON '97. IBM Press, 1997, pp. 15–.

[27] J. Magee, N. Dulay, S. Eisenbach, and J. Kramer, "Specifying distributed software architectures," in *ESEC*, ser. LNCS, W. Schäfer and P. Botella, Eds. Springer Berlin / Heidelberg, 1995, vol. 989, pp. 137–153.

[28] N. Medvidovic and R. Taylor, "A classification and comparison framework for software architecture description languages," *IEEE TSE*, vol. 26, no. 1, pp. 70 –93, Jan 2000.

[29] G. Molter, "Integrating SAAM in domain-centric and reuse-based development processes," in *NOSA*, 1999, pp. 1–10.

[30] R. Monroe, A. Kompanek, R. Melton, and D. Garlan, "Architectural styles, design patterns, and objects," *IEEE Softw.*, vol. 14, no. 1, pp. 43 –52, Jan/Feb 1997.

[31] L. O'Brien Lero, P. Merson, and L. Bass, "Quality attributes for service-oriented architectures," in *SDSOA*, May 2007, pp. 3–3.

[32] I. Ozkaya, L. Bass, R. Nord, and R. Sangwan, "Making practical use of quality attribute information," *IEEE Softw.*, vol. 25, no. 2, pp. 25–33, 2008.

[33] D. E. Perry and A. L. Wolf, "Foundations for the study of software architecture," *SIGSOFT SEN*, vol. 17, pp. 40–52, Oct 1992.

[34] S. Sarkar, A. C. Kak, and G. M. Rama, "Metrics for measuring the quality of modularization of large-scale object-oriented software," *IEEE TSE*, vol. 34, no. 5, pp. 700–720, 2008.

[35] M. Shaw and P. Clements, "The golden age of software architecture," *IEEE Softw.*, vol. 23, no. 2, pp. 31 – 39, Mar/Apr 2006.

[36] B. Tekinerdogan, H. Sozer, and M. Aksit, "Software architecture reliability analysis using failure scenarios," *JSS*, vol. 81, no. 4, pp. 558 – 575, 2008.

[37] L. G. Williams and C. U. Smith, "PASASM: a method for the performance assessment of software architectures," in *IWSP*, ser. WOSP '02. ACM, 2002, pp. 179–189.

[38] P. Clements, R. Kazman, and M. Klein, *Evaluating Software Architectures: Methods and Case Studies*. Addison-Wesley, 2001.

[39] R. Wille, "Formal concept analysis as mathematical theory of concepts and concept hierarchies," in *FCA*, ser. LNCS, B. Ganter, G. Stumme, and R. Wille, Eds., vol. 3626. Springer, 2005, pp. 1–33.

[40] B. Kitchenham, "Evidence-based software engineering and systematic literature reviews," in *PROFES*, ser. Lecture Notes in Computer Science, J. Münch and M. Vierimaa, Eds., vol. 4034. Springer, 2006, p. 3.

[41] J. Poelmans, P. Elzinga, S. Viaene, and G. Dedene, "Formal concept analysis in knowledge discovery: A survey," in *ICCS*, ser. LNCS, M. Croitoru, S. Ferré, and D. Lukose, Eds., vol. 6208. Springer, 2010, pp. 139–153.

Evidence for the Pareto principle in Open Source Software Activity

Mathieu Goeminne and Tom Mens
Institut d'Informatique, Faculté des Sciences
Université de Mons – UMONS
Mons, Belgium
{ mathieu.goeminne | tom.mens }@umons.ac.be

***Abstract*—Numerous empirical studies analyse evolving open source software (OSS) projects, and try to estimate the activity and effort in these projects. Most of these studies, however, only focus on a limited set of artefacts, being source code and defect data. In our research, we extend the analysis by also taking into account mailing list information. The main goal of this article is to find evidence for the Pareto principle in this context, by studying how the activity of developers and users involved in OSS projects is distributed: it appears that most of the activity is carried out by a small group of people. Following the GQM paradigm, we provide evidence for this principle. We selected a range of metrics used in economy to measure inequality in distribution of wealth, and adapted these metrics to assess how OSS project activity is distributed. Regardless of whether we analyse version repositories, bug trackers, or mailing lists, and for all three projects we studied, it turns out that the distribution of activity is highly imbalanced.**

***Index Terms*—software evolution, activity, software project, data mining, empirical study, open source software, GQM, Pareto**

I. Introduction

Numerous empirical studies aim to understand and model how open source software (OSS) evolves over time [1]. In order to gain a deeper understanding of this evolution, it is essential to study not only the software artefacts that evolve (e.g. source code, bug reports, and so on), but also their interplay with the different project members (mainly developers and users) that communicate (e.g., via mailing lists) and collaborate in order to construct and evolve the software.

In this article, we wish to understand how activity is spread over the different members of an OSS project, and how this activity distribution evolves over time. Our hypothesis is that the distribution of activity follows the Pareto principle, in the sense that there is a small group of key persons that carry out most of the activity, regardless of the type of considered activity. To verify this hypothesis, we carry out an empirical study based on the GQM paradigm [2]. We rely on concepts borrowed from econometrics (the use of measurement in economy), and apply them to the field of OSS evolution. In particular, we apply indices that have been introduced for measuring distribution (and inequality) of wealth, and use them to measure the distribution of activity in software development.

The remainder of this paper is structured as follows. Section II explains the methodology we followed and defines the metrics that we rely upon. Section III presents the experimental setup of our empirical study that we have carried out. Section IV presents the results of our analysis of activity distribution in three OSS projects. Section V discusses the evidence we found for the Pareto principle. Section VI presents related work, and Section VII concludes.

II. Methodology

A. GQM paradigm

To gain a deeper understanding of how OSS projects evolve, we follow the well-known **Goal-Question-Metric** (GQM) paradigm. Our main research **Goal** is to understand how activity is distributed over the different stakeholders (developers and users) involved in OSS projects. Once we have gained deeper insight in this issue, we will be able to exploit it to provide dedicated tool support to the OSS community, e.g., by helping newcomers to understand how the community is structured, by improving the way in which the community members communicate and collaborate, by trying to reduce the potential risk of the so-called *bus factor*[1], and so on.

To reach the aforementioned research goal, we raise the following research **Questions**:

1) Is there a core group of OSS project members (developers and/or users) that are significantly more active than the other members?
2) How does the distribution of activity within an OSS community evolve over time?
3) Is there an overlap between the different types of activity (e.g., committing, mailing, submitting and changing bug reports) the community members contribute to?
4) How does the distribution of activity vary across different OSS projects?

As a third step, we need to select appropriate **Metrics** that will enable us to provide a satisfactory answer to each of the above research questions. For our empirical study, we will make use of *basic metrics* to compute the activity of OSS project members, and *aggregate metrics* that allow us to compare these basic metric values across members (to understand how activity is distributed), over time (to understand how they

[1]The *bus factor* refers to the total number of key persons (involved in the project) that would, if they were to be hit by a bus, lead the project into serious problems

evolve), and across projects (to compare the situation between different OSS projects).

B. Basic metrics

To obtain the *basic metrics* of OSS activity, we will extract information from three different types of data sources we have at our disposal: version repositories, mailing lists, and bug trackers. For each of these data sources, we can define metrics that extract and reflect a particular type of activity:

- **Development activity**: the activity of developers committing source code to a version repository, measured as number of commits.
- **Mailing activity**: the activity of project members posting messages to a mailing list, measured as number of mails.
- **Bug tracker activity**: the activity of persons interacting with a bug tracker, measured in three different ways: number of new bug report submissions, number of comments added to existing bug reports, number of changes to existing bug reports.

Since we are not only interested in a static view of a particular snapshot of an OSS project at a particular moment in time, we will extract each of the above activity metrics during the entire life of the considered OSS projects.

C. Aggregate metrics

Since several of the research questions require a comparison of the basic metrics (across persons, across projects, and over time), we need *aggregate metrics* that combine the basic metrics. This is valuable, in particular, if we want to reason about the distribution of activity across OSS project members.

To study such distribution, we borrow ideas from *econometrics*. This discipline uses statistics and metrics to analyse economic data. As an example, various aggregation measures of statistical dispersion have been proposed (e.g., the Hoover, Gini, and Theil indices) and applied to assess the inequality of the wealth distribution among people, regions, countries, and so on.

Recently, some of these aggregation measures have been used for analysing evolving software systems. Vasa et al. [3] proposed to use the Gini index as an alternative to traditional software metrics. Serebrenik et al. [4] proposed to use the Theil index instead. Following this emerging trend, we will use three different aggregation measures to study OSS activity distribution. Below we provide the definitions of the three aggregation measures we selected: the Hoover index, the Gini index, and the Theil index. These definitions rely on two auxiliary definitions.

Let $X = \{x_1, \ldots, x_n\}$ be a set of values indexed in ascending order ($\forall i \in 1 \ldots n-1, x_i \leq x_{i+1}$). The sum of all these values will be called x_{total} (Equation 1). The mean of all values will be called $\bar{x}$ (Equation 2).

$$x_{total} = \sum_{i=1}^{n} x_i \tag{1}$$

$$\bar{x} = \frac{x_{total}}{n} \tag{2}$$

The Hoover index, defined in Equation 3, is one of the simplest ways to assess inequality of wealth or income. Its value is the ratio of incomes to take up from the richest part of the population to redistribute to the poorest one so that the incomes become perfectly equal. A Hoover index of 0 represents perfect equality, while a value of 1 represents perfect inequality.

$$H(X) = \frac{1}{2} \sum_{i=1}^{n} \left| \frac{x_i}{x_{total}} - \frac{1}{n} \right| \tag{3}$$

The Gini index, defined in Equation 4, provides a more complex (but also more representative) way to assess inequality of income. The cumulative function of income distribution is represented by a perfect diagonal if all entities in the population would have the same income, and by a curve under the perfect line otherwise. The Gini index is the value of the surface area between the perfect line and the curve, divided by the surface area below the perfect line.

$$G(X) = 1 - \frac{2}{n-1} \cdot \left(n - \frac{\sum_{i=1}^{n} i x_i}{\sum_{i=1}^{n} x_i} \right) \tag{4}$$

Yet another index to assess inequality of income or wealth is the Theil index. It is based on the Shannon entropy [5], and is defined in Equation 5. Because the Theil index is not bounded *a priori*, one cannot easily compare it with the two aforementioned indices. To normalize the Theil index so that it always returns a value between 0 and 1, we can apply the normalisation function N described in Equation 6 to it.

$$T(X) = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{x_i}{\bar{x}} \cdot \ln \left(\frac{x_i}{\bar{x}} \right) \right) \tag{5}$$

$$N : t \to 1 - e^{-t} \tag{6}$$

III. Experimental setup

A. Implementation

In order to obtain replicable and verifiable results, we do not only need a good methodology. At least as important is automated tool support that enables us to extract data from the different data sources, compute the basic and aggregate metrics based on this data, statistically analyse the obtained metrics, and visually confirm the results.

To this extent, we use and extend our generic framework for analysing open source software projects. This framework, presented in [6], has been developed in a modular and extensible way, facilitating the implementation of new modules meeting specific needs. Support for automatic extraction of data from version repositories, mailing lists and bug tracking systems was already built-in. The framework also supports generation and visualisation of various types of software (project) metrics.

For the specific purposes of this article, we added a new module to compute activity distribution (i.e. the relative activity of each involved person), and different variants of activity are supported. In particular, we implemented the three types of activity defined in Section II-B. We also added a module for computing aggregation indices. Among others, the Hoover, Gini, and Theil index are currently supported. The existing statistic analysis and visualisation modules can directly exploit the information computed by the activity distribution module and the aggregation index module to produce statistical output representing the inequality indices.

B. Pareto principle

Many types of distributions in which people are involved correspond to the so-called *Pareto principle*: roughly 80% of the effects stem from approximately 20% of the causes [7]. This principle and the associated law have been observed repeatedly in a variety of domains, including software evolution [8], [9].

Answering the first research question of Section II-A boils down to finding empirical evidence for the *Pareto principle* in OSS project activity distribution. One should note the difference between the Pareto principle and the related notion of Pareto distribution [10]. While a Pareto distribution satisfies the Pareto principle, the inverse is not true: a statistical distribution may satisfy the Pareto principle without being a Pareto distribution. In fact, many types of *power law* probability distributions have been observed when analysing human activity, and OSS project activity in particular, and the Pareto distribution is only one them [7], [11]. Many power law distributions satisfy the Pareto principle without being a Pareto distribution.

The second research question of Section II-A corresponds to determining whether the Pareto principle is present throughout the entire life of the project, and whether it emerges, stabilises or disappears over time.

C. Selected projects

We will analyse the distribution and evolution of activity on the following OSS projects: Brasero (projects.gnome.org/brasero), Wine (www.winehq.org), and Evince (projects.gnome.org/evince). They have been selected based on a variety of factors: popularity, age size, availability of the necessary data sources for analysis, and so on. Some of the characteristics of the three selected projects are presented in Table I.

OSS project	Brasero	Evince	Wine
main programming language	C	C/C++	C
versioning system	git	svn	git
age (in years)	8	11	11
size (in KLOC)	107	580	2001
# of commits	4100	4000	74500
# of mails	460	1800	14000
# bug reports	250	950	3300
# commiters	206	204	1229
# mailers	102	610	6879
# bug reporters	386	961	2676

TABLE I
MAIN CHARACTERISTICS OF SELECTED OSS PROJECTS. THE REPORTED VALUES HAVE BEEN COMPUTED FOR THE LAST VERSION, NOVEMBER 2010.

IV. EMPIRICAL STUDY

A. Brasero

Figure 1 shows the cumulative activity distribution in the Brasero community, for three types of activity: commits made to the version repository, mails sent to the mailing list, and changes made to bug reports in the bug tracker. These distributions are shown for the last version of Brasero we analysed, namely the one available on November 2010. For the previous versions, with the exception of the earliest versions, we get similar results.

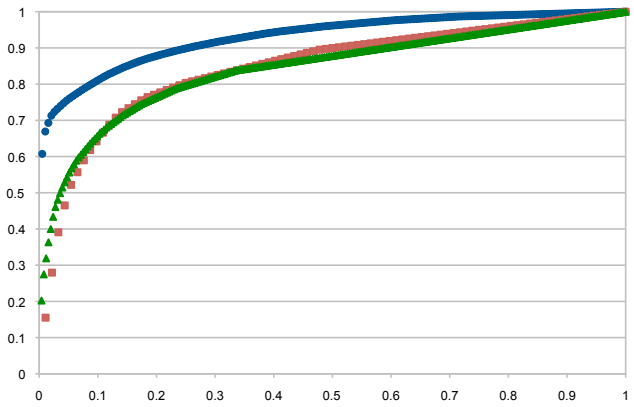


Fig. 1. Cumulative view of distribution of activity for Brasero (November 2010). The x-axis shows the cumulative percentage of active persons, ordered from most to least active; the y-axis shows the cumulative percentage of activity. The upper distribution (blue circles) corresponds to the commit activity . The distribution with red squares corresponds to the mail activity. The distribution with green triangles corresponds to the bug report change activity.

These distributions illustrate that there always is a small core team of persons that account for most of the activity. For the commit activity, 3 out of 193 persons carry out about 70% of the total number of commits. Even more striking is the fact that a single developer accounts for 60% of the total number of commits. For the mail activity, 7 out of 92 persons sent about 60% of all the mails. For the bug report change activity, 5 out of 253 persons carry out about 40% of all bug report changes.

While a detailed statistical analysis of the exact type of distribution is left for future work, we do find clear support for the Pareto principle: for the commit activity, 20% of the most active committers contribute to about 85% of all commits. For the mail (resp. bug report change) activity, 20% of the most active committers contribute with about 75% of all mails (resp. bug report changes).

Figure 2 compares the activity distribution between different types of bug report activities available in the bug tracker: submitting new bug reports, changing existing bug reports, and commenting on existing bug reports. It provides similar

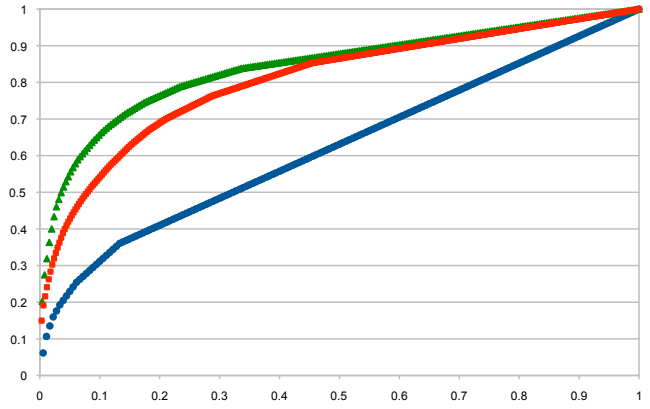


Fig. 2. Cumulative view of distribution of activity for Brasero bug reports (November 2010). The x-axis shows the cumulative percentage of active bug reporters, ordered from most to least active; the y-axis shows the cumulative percentage of bug report activity. The upper distribution (green triangles) corresponds to bug report changes, the middle one (red squares) to bug report submissions, and the lower one (blue circles) to bug report comments.

distributions as those found in Figure 1.

To answer research question 3 of Section II-A, we determined the overlap between different categories of activities (committing, mailing, changing bug reports), by analysing those persons that were involved in different activities. Figure 3 presents the results of this analysis. The triplet notation $(c\%, m\%, b\%)$ used in each of the intersections corresponds to the percentage of activity of a particular individual that contributed to more than one activity category. For example, there is a person with $(61\%, 11\%, 20\%)$, indicating that he contributed to 61% of the total activity of the top 20 most active committers, to 11% of the mail activity of the top 20 most active mailers, and to 20% of the bug report change activity of the top 20 most active bug report changers.

As expected, we find a clear overlap of activity. The two most active committers (out of the top 20) are also very active mailers and bug report changers. In fact, the same two persons account for 67% of the top 20 commit activity, 34% of the top 20 mail activity and 27% of the top 20 bug report change activity in Brasero. We also observe that three of the 20 most active mailers are also active as top 20 bug report changers.

Note that the analysis process for obtaining the results in Figure 3 was manual, which explains the restriction to the top 20 most active individuals only. The reason is that it is quite challenging to automate a reliable identification of identities (logins, e-mail addresses, names) that correspond to the same person. As can be seen in Figure 3, the total sum of all contributors per activity category is 19 instead of 20 for committers and mailers. The reason for this is that, during the manual analysis, we observed that two different identities actually corresponded to the same individual (because he has used two different e-mail addresses or logins over time). Therefore, we merged the corresponding data into a single entity.

To find out how the distribution of activity evolves over

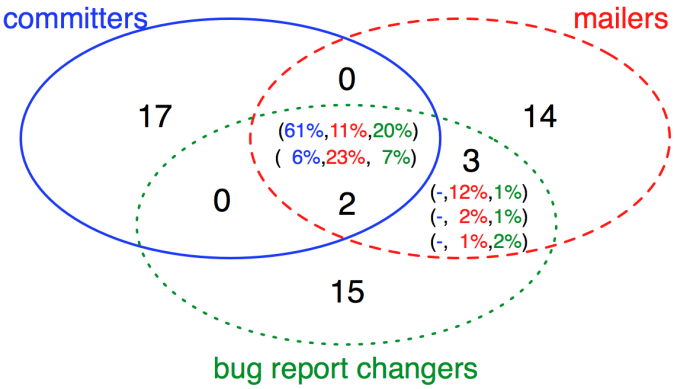


Fig. 3. Overlaps of activity for the top 20 most active individuals of the Brasero community for the three considered categories of activity (November 2010).

time, we used the econometric aggregation measures introduced in Section II-C. Figure 4 displays the evolution of three indices (Hoover, Gini and Theil) for the commits in Brasero. Each data point in this figure corresponds to a different distribution such as the ones shown in Figure 1. We observe that, regardless of the index used, the values do not fluctuate a lot, and tend to stabilise over time. For Gini, for example, we see that the index remains most of the time between 0.8 and 0.9, indicating a very unequal distribution of commit activity for all observed versions. This corroborates what we already observed before: a low number of individuals contribute most of the commits.

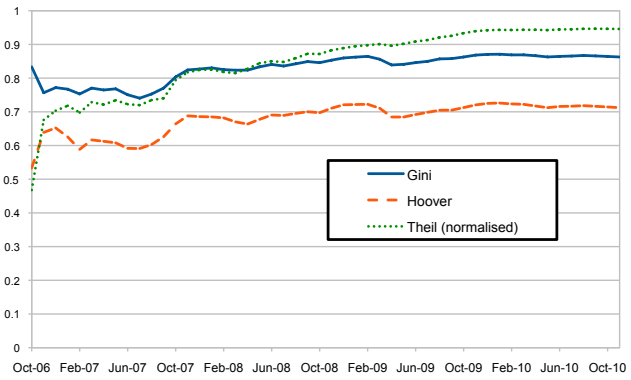


Fig. 4. Comparison of three aggregation indices, Gini (blue straight line), Theil (green dotted line) and Hoover (red dashed line), applied to the evolution of commit activity for Brasero since October 2006.

Figure 5 shows how the Gini index differs across the different activity categories we analysed over time for Brasero: commits, mails and bug report changes. Again, the results correspond to what we observed in the distributions of Figure 1. In all cases, the activity is unequally distributed across individuals. This is especially the case for the commits (with a single committer accounting for 60% of the total number of commits), explaining the high value of the Gini index. For

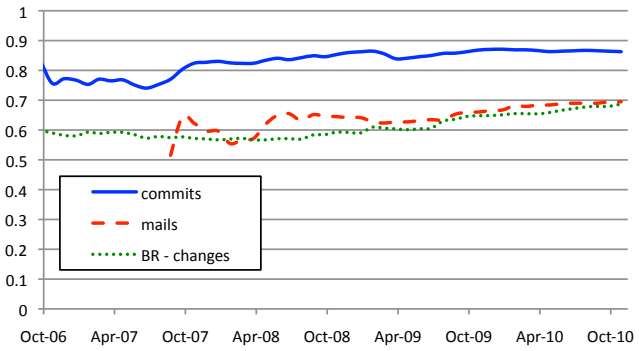


Fig. 5. Comparison of Gini indices over the evolution of Brasero commits (blue straight line) and bug report changes (green dotted line) since October 2006, and mails (red dashed line) since September 2007.

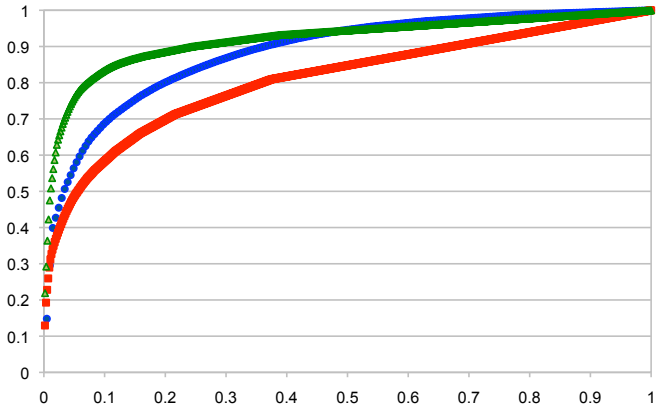


Fig. 6. Cumulative distribution of activity for Evince (November 2010). The distribution with blue circles corresponds to the commit activity. The distribution with red squares corresponds to the mail activity. The distribution with green triangles corresponds to the bug report change activity.

mail activity and bug report change activity, there is also an unequal distribution, but less flagrant than for the commits. This explains why their Gini index curve is below the one for the commit activity.

B. Evince

A study of the evolution of the Evince community provides similar results. Perhaps an important difference is that the development activity is a bit more equally distributed than for Brasero (where we found a single person responsible for 60% of the total commit activity).

Figure 6 shows the cumulative activity distribution for three types of activity for Evince: commits to the version repository, mails sent to the mailing list, and changes made to bug reports in the bug tracker. Evince has two top committers, each accounting for 15% of the total commit activity. The Pareto principle is also clearly present in Figure 6: 20% of all committers contribute to 80% of the total commit activity, 20% of all mailers contribute to 70% of the total mail activity, and 20% of all bug report changers contribute to 88% of the total bug report change activity.

Figure 7 provides a different view on the same data, obtained by manually analysing the top 20 of most active persons for each activity category. As for Brasero, the 4 most active persons of the Evince community contribute to each of the three activity categories. Together, they account for 35% of all top 20 commits, 26% of all top 20 mails, and 36% of all top 20 bug report changes.

Figure 8 shows the evolution of the three econometric aggregation indices on the evolution of Evince's commit activity. After a steep startup phase, we see that the indices start to stabilise rapidly to a more or less stable value. This value is lower than for Brasero, since the distribution of commit activity is a bit more equally distributed for Evince.

Since Figure 8 reveals a similar pattern for all three indices, in Figure 9 we restricted ourselves to the Gini index only. We used it to compare the evolution of the commit, mail and bug report activity of Evince. For each of these activities, we observe that the Gini index stabilises rapidly to a more or less

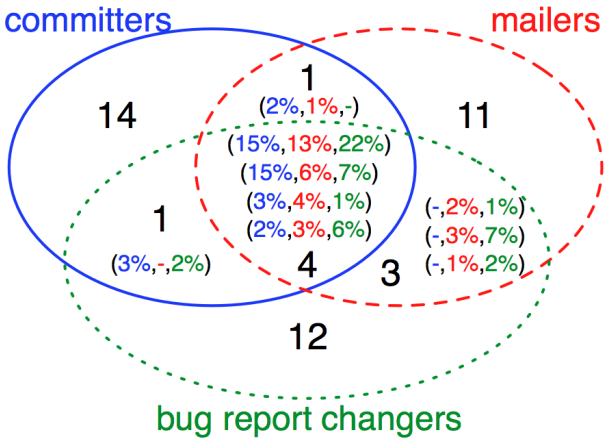


Fig. 7. Overlaps of activity for the top 20 individuals contributing to the considered Evince activity categories (November 2010).

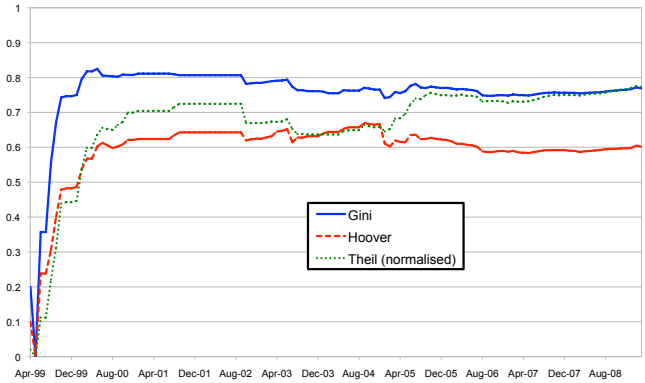


Fig. 8. Comparison of three aggregation indices applied to the evolution of commit activity for Evince since April 1999.

constant value, indicating that the way in which the community is structured is fairly stable. For commits and mails, the high Gini index indicates that the activity is not equally distributed over the community members. For mails, the Gini index is lower so this activity is more spread over different persons.

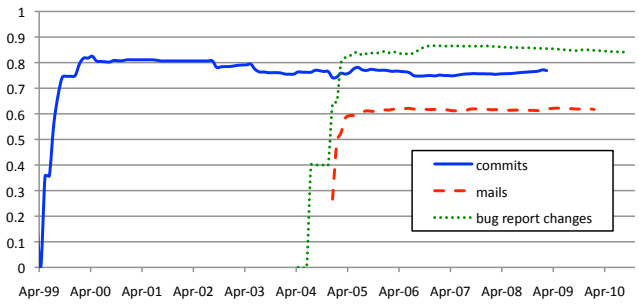


Fig. 9. Comparison of the evolution of Gini indices for Evince, since April 1999 for commits (blue straight line), since January 2005 for mails (red dashed line), and since August 2004 for bug report changes (green dotted line).

C. Wine

A study of the Wine community reveals that the number of persons involved is much bigger than for the other two systems studied. This was already apparent from Table I.

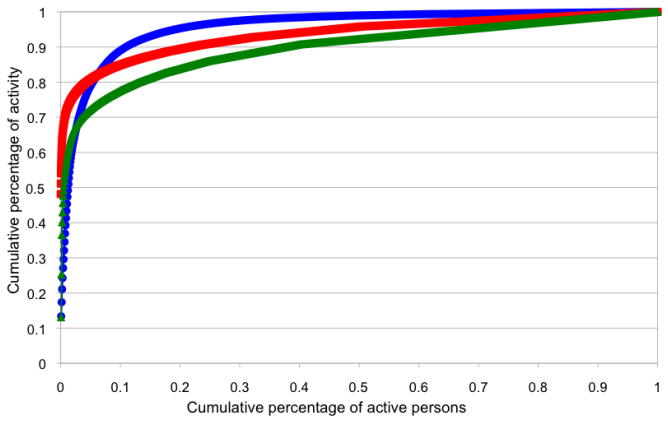


Fig. 10. Cumulative distribution of activity for Wine (November 2010). The distribution with blue circles corresponds to the commit activity. The distribution with red squares corresponds to the mail activity. The distribution with green triangles corresponds to the bug report change activity.

Figure 10 shows the cumulative activity distribution for Wine committers, mailers and bug report changers. The most active committer accounts for 13% of the total commit activity, the two most active bug report changers each account for 13% and 11% of the total change activity, respectively.

Concerning the mail activity of Wine, we observed that there is one huge mailer `wineforum-user@winehq.org` accounting for 48% of the total project mail activity. It turned out that this mailer was in fact an automated transcription of the discussion forum in the mail system. After excluding this outlier from the mail activity data set, we found that the most active mailer only accounts for 4% of the total mail activity.

Figure 10 also provides evidence for the Pareto principle. 11% of all committers account for a total of 90% of commits. Similarly, a total of 13% of bug report changers account for 80% of all changes.

As a side note, we observed that the use of logins and accounts in Wine was poorly structured. For example, many committers have 4 or 5 email addresses that are rarely used. This may be explained by the fact that the Wine community is very open: it is very easy for new persons to become active in this community.

Figure 11 shows the overlap of activity of the top 20 most active persons in each activity category for Wine. Again, we needed to merge two different identities corresponding to the same individual into a single identity (explaining a total sum of 19 instead of 20 for committers). In contrast to the previously analysed projects, none of the 20 most active persons contributed to three different activity categories. Another major difference was that the core group of active persons for Wine was significantly bigger, explaining the smaller percentages we obtained for the most active persons involved in a particular activity.

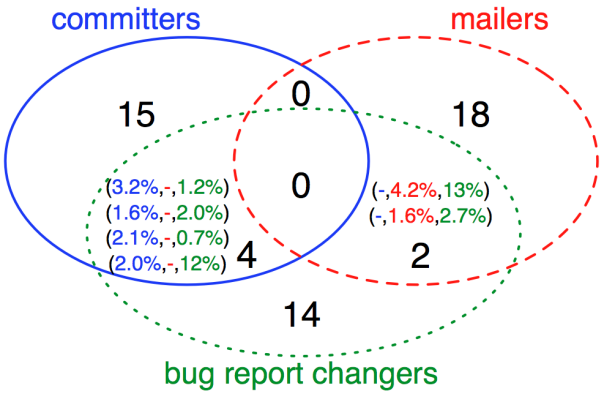


Fig. 11. Overlaps of activity of top 20 individuals contributing to the considered Wine activity categories (November 2010).

Figure 12 compares the evolution of Wine's commit activity using three different aggregation indices. We observe that the Gini and Theil indices are very high and continue to increase over time, indicating a very unequal distribution of activity, with a large group of inactive persons and a small group of active persons.

Figure 13 displays the evolution of the three types of activity for Wine over time, using the Gini index as aggregation measure. For mail activity, the Gini index is initially much lower than for the commit activity, but after a while its value starts to increase to comparable values. This high value is largely explained by the presence of a single artificial mailer (corresponding to the wineforum). For the bug report change activity, we also observe an increasing growth of the Gini index, revealing an increasing inequality of activity distribution over time.

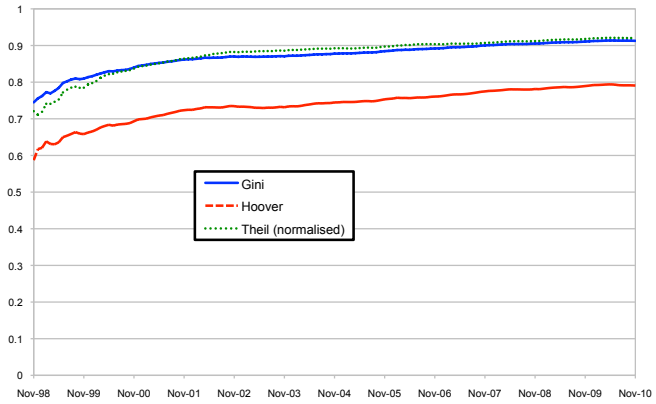


Fig. 12. Comparison of three aggregation indices applied to the evolution of commit activity for Wine since November 1998.

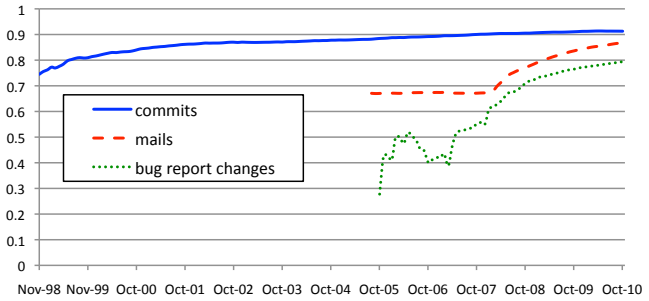


Fig. 13. Comparison of the evolution of Gini indices for Wine, since November 1998 for commits (blue straight line), since September 2005 for mails (red dashed line), and since November 2005 for bug report changes (green dotted line).

V. Discussion

The results shown in the previous section provide strong evidence for the Pareto principle. The activity of contributors to open source projects is not equally distributed: in all three studied projects, a core group of persons appears to carry out the majority of the work. This kind of behaviour may be related to the way in which open source developer communities are structured. According to [12], a typical structure is the so-called onion model. It is followed, among others, by the community in charge of the Linux kernel. In such a layered model, there is a single responsible of the project, surrounded by a small core team of software developers, around which there is a bigger layer of active developers, followed an even bigger layer of occasional developers. The last layer constitutes those users of the project that do not contribute anything themselves. The more to the center of the onion, the more active a developer, and conversely. Although many variants of this layered model exist, the general idea behind it remains the same.

Our empirical analysis showed the usefulness of applying results from econometry to the analysis of the activity in software ecosystems. We used three different aggregation indices of statistical dispersion, the Hoover index, Theil index and Gini index. They all gave similar results, i.e., they tend to evolve in the same way for a given OSS project's history. This probably implies that one can freely choose any of these indices to assess the evolution of activity distribution. This corresponds to the findings of Vasilescu et al. [13] that compared different aggregation measures applied to software metrics and observed a strong correlation between them.

For all three OSS projects we studied, the distribution of activity was initially more equally distributed, but over time the activity tends to become concentrated in a core group of persons that is significantly more active than the others. This knowledge is quite important, as the sudden disappearance of some members of the core group may have an important impact on the future of the software project. In other ways, we found empirical evidence of the so-called *bus factor*, the total number of key persons that would, if they were to be hit by a bus, lead the project into serious problems. Note that this was less the case for Wine, by far the biggest of the three projects, where the activity was more equally distributed over the most active committers than for the other two projects.

For all types of activities, we found a long tail of persons whose activity rate can be largely neglected: during the entire lifetime of the project, they contributed once or twice to one of the considered project activity categories (commits, mails and bug reporting). We also observed that, except for Wine, the most active project members take part in all these activities.

As a potential threat to validity, during our experiments, we encountered some problems to determine which persons contribute to different activity categories (Figures 3, 7, 11). This was mainly a manual and error-prone process that took a lot of effort. In the future we intend to automate this process. To achieve this, an efficient identity merging algorithm is needed that allows one to identify matches between entities participating in different data sources. But for poorly structured projects such as Wine this may still be quite problematic and difficult to automate. We have studied this topic in more detail in [14].

VI. Related Work

Some researchers focus on the study of core teams [15], [16] and the inequality of distribution in software development [13], [3], [4]. Our work distinguishes itself from that of most other researchers involved in mining software repositories [17], [18], [19], [20], [21], who tend to focus on the analysis of the software development *artefacts* (e.g. source code, bug reports, and so on) and the dependencies between those. Instead, our main interest goes to the *individuals* involved in creating and modifying those artefacts, as well as the interaction and communication between those individuals.

Because the data related to software development activity is dispersed and incomplete, and due to the heterogeneity of data formats used in software development, an important effort must be made in order to collect and represent all available data in a uniform way [22], [23]. In previous work [14], we discuss the need to merge identities (logins, e-mail addresses,

names) across different data sources, and compare existing identity merging algorithms that try to achieve this.

Numerous studies have found evidence for the Pareto principle (or, more generally, a power law distribution) in human-related networks [24]. For instance, evidence for a power law distribution has been found in the number of citations in papers [25], the number of sexual partners in human societies [26], and many more. In software evolution, Herraiz founds a double Pareto distribution in software size [8] (using different measures of size). Mitzenmacher generalized this observation for file system distributions [27]. Hunt and Johnson demonstrated [28] that most of the data available in Sourceforge, a software forge for free/open source software[2], follows a Pareto distribution.

VII. Conclusion

In this article, we studied and compared the evolution of OSS project activity. Following the GQM paradigm, our main research goal was to understand how activity is distributed in OSS projects over time. We considered three categories of activity: committing data and code, sending mail and changing bug reports. We extracted and analysed such activity based on three different types of data sources: version repositories, mailing lists, and bug tracking data. We carried out an empirical analysis over three different long-lived OSS projects for which this data was available: Brasero, Evince and Wine.

For all three studied projects and for all considered activity categories, we found evidence for the Pareto principle. The activity distributions showed a strong inequality in the activity of different persons involved in an OSS project: there is a small group of very active members, and a much bigger group of largely inactive members. For two of the three studied projects, the core group of most active members takes part in more than one activity category. In Wine this was much less the case.

In order to gain understanding in how OSS projects evolve, we studied this inequality of distribution over time. To do so we relied on statistical techniques borrowed from econometrics. We applied three economic aggregation measures (the Hoover, Gini and Theil index). The evolution of activity distribution appeared to follow two kinds of behaviour. The first one is typical of a totally new project: at the beginning, the activity is more or less equally distributed, but quickly we observe a tendency towards a more unequal distribution where the activities become more concentrated in a small core team. In the second type of observed behaviour, the activity distribution is already imbalanced since the beginning of the project, and this imbalance continues to become more pronounced over time.

Studying who are the most active persons involved in each type of activity, we discovered that these persons are often very active in different activity categories. For Brasero and Evince, the two projects in which we observed this behaviour, the project is led by a small group of very active members wearing several hats at the same time. We have not yet been able to identify the cause of overlaps between activities, because our definitions for measuring activity need to be refined further.

While Brasero and Evince show a similar evolution of activity distribution and similar overlaps between most active persons' activity categories, the Wine software project appears to have a different behaviour. It has a significantly bigger community, there is significantly less overlap between activity categories, and we observed a higher inequality in the activity distribution. We can only speculate as to the causes of this.

VIII. Future work

While the work presented in this article is very promising, many challenges lay still ahead of us.

We intend to carry out a detailed statistical study of the activity distributions we presented in this article. In particular, we would like to find out which kind of statistical distribution they represent. Can we find evidence for power laws, Pareto distributions or other types of statistical distributions?

We also intend to analyse the activity distribution for subgroups and subprojects, and study how the structure and size of the project community and the software itself affects the type of activity distribution. Another aspect worthy of further study is the evolution of the activity of individuals involved in OSS projects. Can we find evidence for a learning curve for newcomers in a project? Does the activity of core members increase or decrease over time? Can we identify certain typical evolution patterns of activity?

Within an OSS project we would like to study the statistical correlation between different characteristics such as software quality, stakeholder activity, and software size. We also want to study how each of these characteristics correlate across different projects.

The notions of activity studied in this article, and the metrics used for computing them, are perhaps too high-level. We could define and use more fine-grained and more specific definitions, in order to get a more detailed picture of how project members interact, and in order to come up with new effort estimation and effort prediction models based on the activity of project members.

It remains an open question whether the type of activity distribution we observed in our empirical study is specific to open source projects. Our hypothesis is that this is indeed the case. To verify this hypothesis, we would like to repeat our experiment on proprietary, closed source software projects. The main challenge here it to get access to such data, in particular, the evolutionary data related to the activities of the developers involved in these projects.

Finally, the long term goal is to provide assistance or guidance, through interactive or automated tool support, for all stakeholders involved in an (open source) software project. Users may rely on information such as the bus factor to decide on using a particular open source software product. Developers may use it to identify the core developers and influential persons. Project managers may wish to control the activity distribution, to estimate or predict the effort, or to reduce the bus factor risk through a better distribution of activity.

[2]http://sourceforge.net/

ACKNOWLEDGMENT

The research is partially supported by (i) F.R.S.-FNRS FRFC project 2.4515.09 "Research Center on Software Adaptability"; (ii) research project AUWB- 08/12-UMH "Model-Driven Software Evolution", an Action de Recherche Concertée financed by the Ministère de la Communauté française - Direction générale de l'Enseignement non obligatoire et de la Recherche scientifique, Belgium.

REFERENCES

[1] J. Fernandez-Ramil, A. Lozano, M. Wermelinger, and A. Capiluppi, "Empirical studies of open source evolution," in *Software Evolution*, T. Mens and S. Demeyer, Eds. Springer, 2008, pp. 263–288.

[2] V. R. Basili, "Software modeling and measurement: the goal/question/metric paradigm," College Park, MD, USA, Tech. Rep., 1992.

[3] R. Vasa, M. Lumpe, P. Branch, and O. Nierstrasz, "Comparative analysis of evolving software systems using the Gini coefficient," in *Proc. Int'l Conf. Software Maintenance*, 2009, pp. 179–188.

[4] A. Serebrenik and M. van den Brand, "Theil index for aggregation of software metrics values," in *IEEE International Conference on Software Maintenance*. Los Alamitos, CA, USA: IEEE Computer Society, 2010, pp. 1–9.

[5] H. Theil, *Economics and information theory*. Center Math. Stud. Business Econ., Univ. Chicago, 1967.

[6] M. Goeminne and T. Mens, "A framework for analysing and visualising open source software ecosystems," in *Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)*, ser. IWPSE-EVOL '10. New York, NY, USA: ACM, 2010, pp. 42–47.

[7] M. Newman, "Power laws, Pareto distributions and Zipf's law," *Contemporary Physics*, vol. 46, no. 5, pp. 323–351, 2005.

[8] I. Herraiz, "A statistical examination of the evolution and properties of libre software," Ph.D. dissertation, Universidad Rey Juan Carlos, 2008.

[9] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan, "Mining email social networks," in *MSR '06: Proceedings of the 3rd IEEE International Working Conference on Mining Software Repositories*, A. Press, Ed., Shanghai, China, May 2006.

[10] M. Hardy, "Pareto's law," *The Mathematical Intelligencer*, vol. 32, pp. 38–43, 2010, 10.1007/s00283-010-9159-2.

[11] A. Clauset, C. R. Shalizi, and M. E. J. Newman, "Power-law distributions in empirical data," *SIAM Review*, vol. 51, no. 4, pp. 661–703, 2009.

[12] M. Antikainen, T. Aaltonen, and J. Vaisanen, "The role of trust in OSS communities - case Linux Kernel community," in *Open Source Development, Adoption and Innovation*. Springer, Jun. 2007, pp. 223–228.

[13] B. Vasilescu, A. Serebrenik, and M. van den Brand, "Comparative study of software metrics aggregation techniques," in *BENEVOL 2010*, December 2010.

[14] M. Goeminne and T. Mens, "A comparison of identity merging algortihms for open source software ecosystems," *Journal on Systems and Software. [Submitted]*, 2011.

[15] G. Robles, J. M. Gonzalez-Barahona, and I. Herraiz, "Evolution of the core team of developers in libre software projects," in *MSR '09: Proceedings of the 6th IEEE International Working Conference on Mining Software Repositories*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 167–170.

[16] S. Minto and G. C. Murphy, "Recommending emergent teams," in *Proceedings of the Fourth International Workshop on Mining Software Repositories*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 5–.

[17] M. D'Ambros, H. Gall, M. Lanza, and M. Pinzger, "Analysing software repositories to understand software evolution," in *Software Evolution*, T. Mens and S. Demeyer, Eds. Springer, 2008, pp. 37–67.

[18] S. Diehl, H. C. Gall, and A. E. Hassan, Eds., *Special Issue on Mining Software Repositories*, ser. Empirical Software Engineering, vol. 14, no. 3, Jun. 2010.

[19] R. Abreu and R. Premraj, "How developer communication frequency relates to bug introducing changes," in *IWPSE-Evol '09: Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops*. New York, NY, USA: ACM, 2009, pp. 153–158.

[20] D. M. German, "Mining cvs repositories, the softchange experience," in *Proceedings of the First International Workshop on Mining Software Repositories*, Edinburg, Scotland, UK, 2004, pp. 17–21.

[21] W. Poncin, A. Serebrenik, and M. van den Brand, "Process mining software repositories," in *CSMR '11: Proceedings of the European Conference on Software Maintenance and Reengineering.*, 2011.

[22] I. Herraiz, G. Robles, and J. M. Gonzalez-Barahona, "Research friendly software repositories," in *IWPSE-Evol '09: Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops*. New York, NY, USA: ACM, 2009, pp. 19–24.

[23] I. Herraiz, D. Izquierdo-Cortazar, and F. Rivas-Hernández, "Flossmetrics: Free/libre/open source software metrics," in *CSMR '09: Proceedings of the 2009 European Conference on Software Maintenance and Reengineering*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 281–284.

[24] M. E. J. Newman, "The Structure and Function of Complex Networks," *SIAM Review*, vol. 45, no. 2, pp. 167–256, 2003.

[25] S. Redner, "How popular is your paper? An empirical study of the citation distribution," *The European Physical Journal B*, vol. 4, p. 131, 1998.

[26] F. Lilijeros, C. Edling, L. Amaral, E. Stanley, and Y. åberg, "The web of human sexual contacts," *Nature*, vol. 411, pp. 907–908, 2001.

[27] M. Mitzenmacher, "Dynamic models for file sizes and double Pareto distributions," *Internet Mathematics*, vol. 1, pp. 305–333, 2002.

[28] F. Hunt and P. Johnson, "On the Pareto distribution of Open Source projects," in *Proceedings of Open Source Software Development Workshop*, Newcastle, UK, 2002.

Index of Authors