Part 2 : Scientific Information

Main applicant:	Nierstrasz, Oscar
Project title:	Enabling the evolution of J2EE applications through reverse engineering and quality assurance

Contents

1 Summary

Java 2 Platform, Enterprise Edition (J2EE) was first introduced in 1999, and since then it has become one of the standard technologies in enterprise application development. An enterprise application is a complex software product that manipulates much persistent data and interacts with the user through a vast and complex user interface. Like any complex software systems, enterprise applications need to continuously change to adapt to new requirements. However, successive changes lead to a decay in the internal quality unless effort is invested to control it.

To address the complexity of enterprise applications, J2EE offers a conglomerate of several technologies, (*e.g.* Enterprise Java Beans — EJB or Java Server Pages — JSP) using several languages, (*e.g.* Java, XML or SQL). In this context, simply applying existing reverse engineering and quality assurance techniques developed for object-oriented systems fails due to two major reasons: (1) analyzing only the Java source code overlooks the information written in other languages such as the XML configurations, the JSP files, the database structure or the SQL statements, and (2) even when analyzing the Java source code we need to consider the technology specific patterns (*e.g.* implementing specific interfaces).

Very little research effort has been spent on understanding the forces that influence the evolution of J2EE applications. This project aims to conduct a systematic study in reverse engineering and quality assurance of J2EE applications. In particular, we target the following questions each of them being addressed in a separate track:

1. How do we *model* J2EE to support analysis of the different languages?
2. What defines internal quality in J2EE applications and how do we *measure* it?
3. How do we *visualize* the diversity of languages to support understanding of J2EE applications?

J2EE meta-model. A prerequisite to analyzing a system is an explicit meta-model to represent it. Consequently, we want to start by enhancing the meta-model for object-oriented systems with entities that specifically model JSP and EJB. We also intend to build the necessary parsers that will populate the models. We will build on the existing infrastructure for parsing Java, and extend it to include JSP and XML.

J2EE metrics. Classical object-oriented metrics fail to capture the semantics of J2EE. For example, an Entity Bean whose persistency is managed by the container, will be detected as being a Data Class by traditional approaches, because it features mostly data and accessor methods. However, that is precisely the intent behind such Entity Beans, since they are mapped to database entities. In this context, the first problem that we want to tackle is to identify the appropriate metrics and detection strategies. Furthermore, given the diversity of languages, a second problem that we want to tackle is: How do we construct metrics that measure interaction between code written in different languages?

J2EE visualizations. Visualization is not only useful for software reverse engineering, but can also be used as a research tool to help the construction of automatic detection by revealing the results. We aim to construct various visualizations to reveal issues like: the components placed in layers and their relationships, or the application parts that access the database. One general problem we want to address is how to represent the heterogeneous aspects of J2EE applications.

2 Research plan

2.1 State of Research in the Field

We review the research done on analyzing J2EE applications Section 2.1.1. Since only very little research effort has been spent on analyzing J2EE applications, we review the state of the art that has addressed similar problems to those in our context. As the problem of analyzing a new technology like J2EE is similar to that of analyzing object-oriented systems, in Section 2.1.2 we review the work on object-oriented software metrics and quality assurance. Finally, in Section 2.1.3 we review approaches for software visualization.

2.1.1 J2EE patterns and analysis

Prior to the first widely used release of J2EE (*i.e.* version 1.3 in 2001), a series of design patterns was released by Sun [Sun]. The catalogue was also published in the form of a book later the same year [ACM01].

Fowler's patterns for enterprise application architecture are not limited to J2EE, but they are applicable to a wide range of enterprise technology [Fow05]. Although these patterns are presented in an informal manner, they provide an excellent starting point for building automated analyses of J2EE applications.

Marinescu has tackled the problem of improving the classical design flaw detection strategies by taking into account the enterprise applications specific patterns [Mar06]. In particular, she studied the detection of Data Class and Feature Envy design flaws [FBB+99] by also taking into account the interaction with the database.

From a different perspective, Cecchet *et al.* measured the performance and scalability of EJB [CMZ02]. They have analyzed five versions of the same system, each version being implemented using a different design. Among others, they have concluded that the performance of an application based exclusively on Entity Beans depends on whether the container uses reflection or not.

Several tools have been built to provide an integrated environment for the development of J2EE applications (*e.g.* Bea Workshop[1], JBoss Eclipse IDE[2] or other Eclipse plugins[3]). Apart from services like code generation and transparent configuration management, these tools also provide code completion throughout the entire technological space. For example, when editing Java source code inside a JSP file, the methods of the Java class are displayed. However, the primary focus of these tools is on optimization of development rather than on offering support for quality assurance.

2.1.2 Software metrics and quality assurance

Object-oriented design claims to support essential software quality goals, like maintainability and reusability, with mechanisms like encapsulation of data, inheritance and dynamic binding [Mey88]. However, in the late 1990's, the software industry was confronted with a large number of legacy software systems that lacked all of the aforementioned qualities: they were instead inflexible and hard to reuse [Rie96, FBB+99, FP96].

Over the last decades, the demand for quality in software products has become increasingly emphasized. As DeMarco pointed out the only way we can control quality is by measuring it [deM86]. This led to the definition of a large number of metrics that were designed to take the specifics of the object-oriented technology into account [HS96, LK94, LH93, CK94, HM96]. For example, concepts of coupling and cohesion had to be redefined for object-oriented systems [BDW98, BDW99].

Design flaw detection strategies encapsulate known heuristics and consist of rules that combine metrics and thresholds [Mar04, MSG99]. The problem of setting the thresholds has been addressed by Mihancea and Marinescu, by using machine learning to optimize the thresholds against a known

[1] `http://www.bea.com/framework.jsp?CNT=index.htm\&FP=/content/products/workshop/workshop/`
[2] `http://www.jboss.com/products/jbosside`
[3] `http://eclipse-plugins.2y.net/eclipse/plugins.jsp?category=J2EE+development+platform`

training set [MM05]. Another approach to produce reliable thresholds was to analyze a large suite of systems and average the typical values [LM06]. Sahraoui *et al.* proposed the use of fuzzy threshold values instead of fixed values [SBL01].

Metrics represent the building blocks for quality models. Factor-Criteria-Metric Models have long been used for procedural code [MRW76, BBK+78]. With the shift to object-orientation, they have also been adapted to a model called QMOOD [BD02]. Marinescu and Raţiu have criticized the Factor-Criteria-Metric Models for their gap between the problem and the solution. They proposed an alternative quality model, called Factor-Strategy model, that, instead of depending on metrics directly, depends on detection strategies [MR04]. As design flaw detection strategies encapsulate heuristics, this model bridges the gap between the quality problem and the solution to remedy the problem. Trifu *et al.* have used them to propose automated correction strategies [TSG04].

2.1.3 Software visualization

Rather then providing an automatic detection of problems, visualization is concerned with displaying the data in a way that enables the eye to interpret them [SDBP98]. According to a recent survey, most specialists consider visualization "absolutely necessary, or important" for software maintenance, reverse engineering and reengineering [Kos03].

Representing programs as graphs has long been used for software visualization, and several tools have implemented this approach: Hy+ [CM93], SHriMP [SM95, MADSM01], Rigi [Mül86, MK88]. Perhaps the most widely used visual notation in software engineering is that offered by the Unified Modeling Language (UML) [Fow03].

Once a visualization is rendered on the screen, the user not only wants to look at it, he also wants to interact with it. According to Storey *et al.* [SFM99] this helps to reduce the cognitive overhead of any visualization. Furthermore, the user also wants to customize the views. Vizz3D embodies this idea and offers general visualizations that can be customized by the user [PLL05].

Representing programs as graphs is not the only means of representation. SV3D presents lines of code as dots and each dot can be associated with different information such as the nesting level or the control flow [MFM03]. For quantitative information, such as the occurrence of a phenomena, 3D is used. Control Structure Diagrams [HCIM02] are based on the layout of the code since they are intended to support the programmers to understand the flow and structure of methods.

A challenge for visualizing abstract data, such as source code, is to distinguish between the different parts. UML uses text to distinguish between similar entities, but when large amounts of data are presented, text is not a recognizable element. Another possibility is to generate automatically VisualIDs [LRFN04]. VisualIDs have been successfully used in the context of browsing the file system, and we believe it is a promising path to be exploited in software visualization.

References

[ACM01] Deepak Alur, John Crupi, and Dan Malks. *Core J2EE Patterns: Best Practices and Design Strategies*. Pearson Education, 2001.

[BBK+78] B. Boehm, J. Brown, H. Kaspar, M. Lipow, G. McLeod, and M. Merritt. *Characteristics of Software Quality*. North Holland, 1978.

[BD02] Jagdish Bansiya and Carl Davis. A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on Software Engineering*, 28(1):4–17, January 2002.

[BDW98] Lionel C. Briand, John W. Daly, and Jürgen Wüst. A Unified Framework for Cohesion Measurement in Object-Oriented Systems. *Empirical Software Engineering: An International Journal*, 3(1):65–117, 1998.

[BDW99] Lionel C. Briand, John W. Daly, and Jürgen K. Wüst. A Unified Framework for Coupling Measurement in Object-Oriented Systems. *IEEE Transactions on Software Engineering*, 25(1):91–121, 1999.

[CK94] Shyam R. Chidamber and Chris F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, June 1994.

[CM93] Mariano P. Consens and Alberto O. Mendelzon. Hy+: A hygraph-based query and visualisation system. In *Proceeding of the 1993 ACM SIGMOD International Conference on Management Data, SIGMOD Record Volume 22, No. 2*, pages 511–516, 1993.

[CMZ02] Emmanuel Cecchet, Julie Marguerite, and Willy Zwaenepoel. Performance and scalability of EJB applications. In *Proceedings International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'02)*, pages 246–261, 2002.

[deM86] Tom deMarco. *Controlling Software Projects: Management, Measurement, and Estimates*. Springer-Verlag, 1986.

[FBB+99] Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts. *Refactoring: Improving the Design of Existing Code*. Addison Wesley, 1999.

[Fow03] Martin Fowler. *UML Distilled*. Addison Wesley, 2003.

[Fow05] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison Wesley, 2005.

[FP96] Norman Fenton and Shari Lawrence Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. International Thomson Computer Press, London, UK, second edition, 1996.

[HCIM02] Dean Hendrix, James H. Cross II, and Saeed Maghsoodloo. The Effectiveness of Control Structure Diagrams in Source Code Comprehension Activities. *IEEE Transactions on Software Engineering*, 28(5):463–477, May 2002.

[HM96] M. Hitz and B. Montazeri. Chidamber and Kemerer's metrics suite; a measurement theory perspective. *IEEE Transactions on Software Engineering*, 22(4):267–271, April 1996.

[HS96] Brian Henderson-Sellers. *Object-Oriented Metrics: Measures of Complexity*. Prentice-Hall, 1996.

[Kos03] Rainer Koschke. Software visualization in software maintenance, reverse engineering, and re-engineering: a research survey. *Journal of Software Maintenance and Evolution: Research and Practice*, 15(2):87–109, 2003.

[LH93] W. Li and S. Henry. Object oriented metrics that predict maintainability. *Journal of System Software*, 23(2):111–122, 1993.

[LK94] Mark Lorenz and Jeff Kidd. *Object-Oriented Software Metrics: A Practical Guide*. Prentice-Hall, 1994.

[LM06] Michele Lanza and Radu Marinescu. *Object-Oriented Metrics in Practice*. Springer-Verlag, 2006.

[LRFN04] J. Lewis, Ruth Rosenholtz, Nickson Fong, and Ulrich Neumann. VisualIDs: automatic distinctive icons for desktop interfaces. *ACM Transactions on Graphics*, 23(3):416–423, August 2004.

[MADSM01] C. Best M.-A. D. Storey and J. Michaud. SHriMP Views: An interactive and customizable environment for software exploration. In *Proceedings of International Workshop on Program Comprehension (IWPC '2001)*, 2001.

[Mar04] Radu Marinescu. Detection strategies: Metrics-based rules for detecting design flaws. In *20th IEEE International Conference on Software Maintenance (ICSM'04)*, pages 350–359, Los Alamitos CA, 2004. IEEE Computer Society Press.

[Mar06] Cristina Marinescu. Identification of design roles for the assessment of design quality in enterprise applications. In *Proceedings of International Conference on Program Comprehension (ICPC 2006)*, pages 169–180, Los Alamitos CA, 2006. IEEE Computer Society Press.

[Mey88] Bertrand Meyer. *Object-oriented Software Construction*. Prentice-Hall, 1988.

[MFM03] Andrian Marcus, Louis Feng, and Jonathan I. Maletic. 3d representations for software visualization. In *Proceedings of the ACM Symposium on Software Visualization*, pages 27–ff. IEEE, 2003.

[MK88] H. A. Müller and K. Klashinsky. Rigi – a system for programming-in-the-large. In *ICSE '88: Proceedings of the 10th international conference on Software engineering*, pages 80–86. IEEE Computer Society Press, 1988.

[MM05] Petru Mihancea and Radu Marinescu. Towards the optimization of automatic detection of design flaws in object-oriented software systems. In *Proceedings of European Conference on Software Maintenance (CSMR 2005)*, pages 92–101, 2005.

[MR04] Radu Marinescu and Daniel Raţiu. Quantifying the quality of object-oriented design: the factor-strategy model. In *Proceedings 11th Working Conference on Reverse Engineering (WCRE'04)*, pages 192–201, Los Alamitos CA, 2004. IEEE Computer Society Press.

[MRW76] Jim McCall, Paul Richards, and Gene Walters. *Factors in Software Quality.* NTIS Springfield, 1976.

[MSG99] Thierry Miceli, Houari A. Sahraoui, and Robert Godin. A metric based technique for design flaws detection and correction. In *Proceedings IEEE Automated Software Engineering Conference (ASE)*, 1999.

[Mül86] Hausi A. Müller. *Rigi — A Model for Software System Construction, Integration, and Evaluation based on Module Interface Specifications.* PhD thesis, Rice University, 1986.

[PLL05] Thomas Panas, Rüdiger Lincke, and Welf Löwe. Online-configuration of software visualization with Vizz3D. In *Proceedings of ACM Symposium on Software Visualization (SOFTVIS 2005)*, pages 173–182, 2005.

[Rie96] Arthur Riel. *Object-Oriented Design Heuristics.* Addison Wesley, Boston MA, 1996.

[SBL01] Houari Sahraoui, Mounir Boukadoum, and Hakim Lounis. Building quality estimation models with fuzzy threshold values. *L'Objet*, 7(4), December 2001.

[SDBP98] John T. Stasko, John Domingue, Marc H. Brown, and Blaine A. Price, editors. *Software Visualization — Programming as a Multimedia Experience.* The MIT Press, 1998.

[SFM99] Margaret-Anne D. Storey, F. David Fracchia, and Hausi A. Müller. Cognitive design elements to support the construction of a mental model during software exploration. *Journal of Software Systems*, 44:171–185, 1999.

[SM95] Margaret-Anne D. Storey and Hausi A. Müller. Manipulating and documenting software structures using SHriMP Views. In *Proceedings of ICSM '95 (International Conference on Software Maintenance)*, pages 275–284. IEEE Computer Society Press, 1995.

[Sun] Sun j2ee patterns catalogue. http://java.sun.com/blueprints/corej2eepatterns/Patterns-/index.html.

[TSG04] Adrian Trifu, Olaf Seng, and Thomas Genssler. Automated design flaw correction in object-oriented systems. In *Proceedings 8th European Conference on Software Maintenance and Reengineering (CSMR 2004)*, pages 174–183, Los Alamitos CA, 2004. IEEE Computer Society Press.

2.2 Research Fields

The Software Composition Group (SCG) carries out research in programming languages and software engineering methods to support the construction of flexible and open software systems. In recent years, the research has focussed on techniques and mechanisms to support software evolution.

Our ongoing SNF research project[4] is concerned specifically with developing support for software evolution in the programming language and development tools. The work proposed here falls out of the scope of our current SNF project, which is more concerned with programming language technology than with reverse engineering and quality assessment.

Another SNF SCOPES project[5] supports cooperation in the realm of reengineering between research groups in Berne and Lugano, Switzerland and Timisoara, Romania. The SCOPES project is directly relevant to the current proposal, but it should be noted that SCOPES provide Swiss partners only with a modest budget for travel, and does not cover salaries.

This project will be carried out in the context of the Moose analysis environment[6]. Moose was conceived in 1997 in the context of the FAMOOS European project[7], and since then, it has continually evolved to support research[8] in reverse engineering and quality assurance within the Software Composition Group and other research groups [DGLD05, DT03, NDG05, DDN02].

Moose is a reengineering environment that provides a flexible framework on which several analysis tools have been implemented: CodeCrawler is a general purpose visualization tool [LD05], ConAn is a concept analysis tool [Aré05], Chronia is a tool for analyzing CVS repositories [See06], Hapax is a tool to analyze the linguistic information from the source code [KDG05], Mondrian is a scripting visualization engine [Mey06], SmallDude is a tool for duplication detection [BGM06], Softwarenaut is an interactive visualization tool [LLG06], TraceScraper is a tool for analyzing dynamic information [GD05], Van is a tool for evolution analysis [Gîr05].

In this section we review recent SCG publications that are particularly relevant to this proposal. In particular, we summarize the research we have carried out in the following areas: (1) meta-modeling and parsing, (2) software metrics, (3) software visualization.

All cited papers are available in electronic form from the SCG web site:

`http://www.iam.unibe.ch/~scg/cgi-bin/scgpubs.cgi`

2.2.1 Meta-modeling and parsing

Meta-modeling is a primary concern at Software Composition Group, as it provides the basic blocks for building high-level analyses. That is why we actively build meta-models to support our research in reverse engineering.

FAMIX is a unified meta-model developed in the SCG for reengineering object-oriented systems [Tic01]. FAMIX modeled static information like classes, methods, functions *etc.*.

The work on analyzing software evolution has led to the construction of a generic meta-model called Hismo which is based on modeling evolution as a first class entity [Gîr05]. Hismo is generic in that it extends any structural meta-model with the notion of *history* as a sequence of *versions* [GD06]. Recently, we have also participated in a joint effort to produce a unified meta-model and a file format for supporting mining software repositories [KZK+06].

Dynamic analysis reasons about the execution of programs (*i.e.* how methods get invoked at runtime, and how objects are passed around the system). In this context, we have extended the structural entities of FAMIX with execution entities such as trace and activation. The resulting meta-model is called Dynamix [Gre07].

[4] SNF project # 200020-113342: Analyzing, Capturing and Taming Software Change, Oct 1, 2006 - Sept. 30, 2008. See `http://www.iam.unibe.ch/~scg/Research/SNF06/`

[5] SNF SCOPES/JRP Project IB7320-110997: Network Of Reengineering Expertise (NOREX), 2005-2007. See `http://www.iam.unibe.ch/~scg/Research/NOREX/`.

[6] `http://moose.unibe.ch`

[7] ESPRIT Project 21975: Framework-based Approach for Mastering Object-Oriented Software Evolution. Sept. 1996-Sept. 1999.

[8] SNF project 620-066077: Recast: Evolution of Object-Oriented Applications. 2002-2006.

To analyze an unknown system we need parsers to produce models that conform to the wanted meta-model. However, when facing a new language, the parser might not be readily available.That is why we have built an iterative technique to parse unknown languages based on provided examples of mappings of the code to entities in the meta-model [Kob05, NKG+07]. Using the mapping examples, we automatically produce a grammar that we check against the rest of the files of the system.

JSP poses a challenge for parsing, as it is embedded in HTML. We initiated an open-source project for a JSP parser that is a plugin of Eclipse. It currently supports standard JSP code, but it ignores custom tags. In the future, we plan to extend the parser to deal with all the details of JSP [Gur06].

We have also investigated the use of meta-modeling from a model-driven engineering perspective. Our primary experience comes from the construction of Moose. Currently, Moose owes its generality to an extensive meta-modeling infrastructure based on MOF (Meta-Object Facility) [DG06]. Based on this infrastructure, we have constructed generic import-export capabilities and user interfaces for manipulating models.

2.2.2 Software metrics

In an initial study, we have determined that size and inheritance measurements are not reliable indicators for detecting problems, but are only useful for indicating stability [DD99].

To characterize software packages we have developed a series of simple metrics, and we use a Kiviat diagram to display them [DLP05]. The goal of the diagrams is to give a perspective on the nature of the package from a client/provider perspective.

We have used metrics and metric-based rules to identify what changed in the system over time. In an initial study we have identified refactorings using metric-based rules that are applied to two versions of the system [DDN00].

Recently, we have devised evolution metrics based on the Hismo meta-model that summarize the entire history [Gîr05]. We have defined Yesterday's Weather, a measurement that indicates the relevance of starting the analysis of a new system from the latest changed parts [GDL04]. In another application, we have shown that evolution information can be combined with structural information for to improve the detection rate of design flaws [RDGM04].

To support the construction of metrics, we have contributed to the construction of the SAIL language [MMG05]. SAIL is an interpreted language that offers mechanisms for manipulating groups of objects and for aggregating properties into numerical values.

2.2.3 Software visualization

CodeCrawler is a generic graph visualization tool [Lan03a]. It was born as an implementation of the concept of Polymetric View [LD03]. The Polymetric View is a generic graph visualization that shows nodes as rectangles, and maps up to five metrics mapped on the dimensions (*i.e.* width and height), color and position of the rectangles (*i.e.* x and y). Polymetric Views have been widely used in several areas: showing the hierarchies of the system with System Complexity View [Lan03b], showing class internals with Class Blueprint [DL05], showing code duplication [Rie04], showing dynamic information [DFW04], showing the evolution of classes using the Evolution Matrix [LD02], or showing the evolution of class hierarchies with Hierarchy Evolution View [GLD05].

To show how different properties spread over the software system, we have developed a generic visualization called Distribution Map [DGK06]. A Distribution Map shows partitions of the system (*e.g.* classes grouped in packages) and maps on the color the properties. We have applied this visualization to reveal the spread of topics detected with linguistic analysis [**?**], and to analyze how developers participate in the development of features [GGD07].

A novel technique to analyze dynamic information was augmented with a visualization of the execution traces as signals [KG06]. The traces were ordered to visually determine commonalities between the different executions.

We have also experimented with the use of 3D visualization and animation to identify trace visualization [GLW06]. The animation is used to show a movie of the events to detect execution hot spots.

Mondrian is a scripting visualization engine that allows for the specification of the visualization based on meta-model transformations [MGL06]. The goal of Mondrian is to allow for fast prototyping of visualization. Mondrian received 2nd prize at ESUG Innovation Contest 2006, and is currently used for research in several Universities.

Microprints display source code using horizontal lines to denote each individual source line [DLR05]. Several color schemes are added on top to reveal different parts like assignments to variables, super calls or decision points.

The architecture of a software systems can be hidden at different levels within the package hierarchy. That is why, we have explored the use of interactive navigation to support visual architecture recovery [LLG06].

Code duplication is a universal source of problems in software development. That is why we have developed a language-independent visual approach that uses a correlation matrix to identify the duplications [DRD99]. We have also used a similar correlation matrix to identify implementation topics using information retrieval [KDG05].

Versioning systems record information about the developers that performed the changes. We have devised a metric to detect the owner of a file, and then we have visualized how owners change over time [GKSD05]. Based on the visualization we could detect patterns of developer behavior, as well as the zone of influence for each developer. Furthermore, we have used the information about the author of each line of code to reveal who copies from whom [BGM06]. We have built a visualization and performed an empirical study to identify several types of code clones.

References

[Aré05] Gabriela Arévalo. *High Level Views in Object-Oriented Systems using Formal Concept Analysis*. PhD thesis, University of Berne, Berne, January 2005.

[BGM06] Mihai Balint, Tudor Gîrba, and Radu Marinescu. How developers copy. In *Proceedings of International Conference on Program Comprehension (ICPC 2006)*, pages 56–65, 2006.

[DD99] Serge Demeyer and Stéphane Ducasse. Metrics, do they really help? In Jacques Malenfant, editor, *Proceedings of Languages et Modèles à Objets (LMO'99)*, pages 69–82. HERMES Science Publications, Paris, 1999.

[DDN00] Serge Demeyer, Stéphane Ducasse, and Oscar Nierstrasz. Finding refactorings via change metrics. In *Proceedings of 15th International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '00)*, pages 166–178, New York NY, 2000. ACM Press. Also appeared in ACM SIGPLAN Notices 35 (10).

[DDN02] Serge Demeyer, Stéphane Ducasse, and Oscar Nierstrasz. *Object-Oriented Reengineering Patterns*. Morgan Kaufmann, 2002.

[DFW04] Stéphane Ducasse, Michael Freidig, and Roel Wuyts. Logic and trace-based object-oriented application testing. In *Fifth International Workshop on Object-Oriented Reengineering (WOOR 2004)*, 2004.

[DG06] Stéphane Ducasse and Tudor Gîrba. Using Smalltalk as a reflective executable meta-language. In *International Conference on Model Driven Engineering Languages and Systems (Models/UML 2006)*, volume 4199 of *LNCS*, pages 604–618, Berlin, Germany, 2006. Springer-Verlag.

[DGK06] Stéphane Ducasse, Tudor Gîrba, and Adrian Kuhn. Distribution map. In *Proceedings of 22nd IEEE International Conference on Software Maintenance (ICSM '06)*, pages 203–212, Los Alamitos CA, 2006. IEEE Computer Society.

[DGLD05] Stéphane Ducasse, Tudor Gîrba, Michele Lanza, and Serge Demeyer. Moose: a collaborative and extensible reengineering environment. In *Tools for Software Maintenance and Reengineering*, RCOST / Software Technology Series, pages 55–71. Franco Angeli, Milano, 2005.

[DL05] Stéphane Ducasse and Michele Lanza. The class blueprint: Visually supporting the understanding of classes. *Transactions on Software Engineering (TSE)*, 31(1):75–90, January 2005.

[DLP05] Stéphane Ducasse, Michele Lanza, and Laura Ponisio. Butterflies: A visual approach to characterize packages. In *Proceedings of the 11th IEEE International Software Metrics Symposium (METRICS'05)*, pages 70–77. IEEE Computer Society, 2005.

[DLR05] Stéphane Ducasse, Michele Lanza, and Romain Robbes. Multi-level method understanding using Microprints. In *Proceedings of VISSOFT 2005 (3th IEEE International Workshop on Visualizing Software for Understanding)*, September 2005.

[DRD99] Stéphane Ducasse, Matthias Rieger, and Serge Demeyer. A language independent approach for detecting duplicated code. In Hongji Yang and Lee White, editors, *Proceedings of 15th IEEE International Conference on Software Maintenance (ICSM'99)*, pages 109–118. IEEE Computer Society, September 1999.

[DT03] Stéphane Ducasse and Sander Tichelaar. Dimensions of reengineering environment infrastructures. *Journal of Software Maintenance and Evolution: Research and Practice (JSME)*, 15(5):345–373, October 2003.

[Gîr05] Tudor Gîrba. *Modeling History to Understand Software Evolution*. PhD thesis, University of Berne, Berne, November 2005.

[GD05] Orla Greevy and Stéphane Ducasse. Correlating features and code using a compact two-sided trace analysis approach. In *Proceedings of 9th European Conference on Software Maintenance and Reengineering (CSMR'05)*, pages 314–323, Los Alamitos CA, 2005. IEEE Computer Society.

[GD06] Tudor Gîrba and Stéphane Ducasse. Modeling history to analyze software evolution. *Journal of Software Maintenance: Research and Practice (JSME)*, 18:207–236, 2006.

[GDL04] Tudor Gîrba, Stéphane Ducasse, and Michele Lanza. Yesterday's Weather: Guiding early reverse engineering efforts by summarizing the evolution of changes. In *Proceedings of 20th IEEE International Conference on Software Maintenance (ICSM'04)*, pages 40–49, Los Alamitos CA, September 2004. IEEE Computer Society.

[GGD07] Orla Greevy, Tudor Gîrba, and Stéphane Ducasse. How developers develop features. In *Proceedings of 11th European Conference on Software Maintenance and Reengineering (CSMR 2007)*, pages 256–274, Los Alamitos CA, 2007. IEEE Computer Society.

[GKSD05] Tudor Gîrba, Adrian Kuhn, Mauricio Seeberger, and Stéphane Ducasse. How developers drive software evolution. In *Proceedings of International Workshop on Principles of Software Evolution (IWPSE 2005)*, pages 113–122. IEEE Computer Society Press, 2005.

[GLD05] Tudor Gîrba, Michele Lanza, and Stéphane Ducasse. Characterizing the evolution of class hierarchies. In *Proceedings of 9th European Conference on Software Maintenance and Reengineering (CSMR'05)*, pages 2–11, Los Alamitos CA, 2005. IEEE Computer Society.

[GLW06] Orla Greevy, Michele Lanza, and Christoph Wysseier. Visualizing live software systems in 3D. In *Proceedings of SoftVis 2006 (ACM Symposium on Software Visualization)*, September 2006.

[Gre07] Orla Greevy. *Enriching Reverse Engineering with Feature Analysis*. PhD thesis, University of Berne, May 2007.

[Gur06] David Gurtner. Importing JSP into Moose. Bachelor's thesis, University of Bern, July 2006.

[KDG05] Adrian Kuhn, Stéphane Ducasse, and Tudor Gîrba. Enriching reverse engineering with semantic clustering. In *Proceedings of 12th Working Conference on Reverse Engineering (WCRE'05)*, pages 113–122, Los Alamitos CA, November 2005. IEEE Computer Society Press.

[KG06] Adrian Kuhn and Orla Greevy. Summarizing traces as signals in time. In *Proceedings IEEE Workshop on Program Comprehension through Dynamic Analysis (PCODA 2006)*, pages 01–06, Los Alamitos CA, October 2006. IEEE Computer Society Press.

[Kob05] Markus Kobel. Parsing by example. Diploma thesis, University of Bern, April 2005.

[KZK+06] Sunghun Kim, Thomas Zimmermann, Miryung Kim, Ahmed Hassan, Audris Mockus, Tudor Gîrba, Martin Pinzger, James Whitehead, and Andreas Zeller. TA-RE: An exchange language for mining software repositories. In *Proceedings Workshop on Mining Software Repositories (MSR 2006)*, pages 22–25, 2006.

[Lan03a] Michele Lanza. Codecrawler — lessons learned in building a software visualization tool. In *Proceedings of CSMR 2003*, pages 409–418. IEEE Press, 2003.

[Lan03b] Michele Lanza. *Object-Oriented Reverse Engineering — Coarse-grained, Fine-grained, and Evolutionary Software Visualization.* PhD thesis, University of Berne, May 2003.

[LD02] Michele Lanza and Stéphane Ducasse. Understanding software evolution using a combination of software visualization and software metrics. In *Proceedings of Langages et Modèles à Objets (LMO'02)*, pages 135–149, Paris, 2002. Lavoisier.

[LD03] Michele Lanza and Stéphane Ducasse. Polymetric views—a lightweight visual approach to reverse engineering. *Transactions on Software Engineering (TSE)*, 29(9):782–795, September 2003.

[LD05] Michele Lanza and Stéphane Ducasse. Codecrawler–an extensible and language independent 2d and 3d software visualization tool. In *Tools for Software Maintenance and Reengineering*, RCOST / Software Technology Series, pages 74–94. Franco Angeli, Milano, 2005.

[LLG06] Mircea Lungu, Michele Lanza, and Tudor Gîrba. Package patterns for visual architecture recovery. In *Proceedings of CSMR 2006 (10th European Conference on Software Maintenance and Reengineering)*, pages 185–196, Los Alamitos CA, 2006. IEEE Computer Society Press.

[Mey06] Michael Meyer. Scripting interactive visualizations. Master's thesis, University of Bern, November 2006.

[MGL06] Michael Meyer, Tudor Gîrba, and Mircea Lungu. Mondrian: An agile visualization framework. In *ACM Symposium on Software Visualization (SoftVis 2006)*, pages 135–144, New York, NY, USA, 2006. ACM Press.

[MMG05] Cristina Marinescu, Radu Marinescu, and Tudor Gîrba. Towards a simplified implementation of object-oriented design metrics. In *METRICS 2005*, pages 110–119, 2005.

[NDG05] Oscar Nierstrasz, Stéphane Ducasse, and Tudor Gîrba. The story of Moose: an agile reengineering environment. In *Proceedings of the European Software Engineering Conference (ESEC/FSE 2005)*, pages 1–10, New York NY, 2005. ACM Press. Invited paper.

[NKG+07] Oscar Nierstrasz, Markus Kobel, Tudor Gîrba, Michele Lanza, and Horst Bunke. Example-driven reconstruction of software models. In *Proceedings of Conference on Software Maintenance and Reengineering (CSMR 2007)*, pages 275–286, Los Alamitos CA, 2007. IEEE Computer Society Press.

[RDGM04] Daniel Raţiu, Stéphane Ducasse, Tudor Gîrba, and Radu Marinescu. Using history information to improve design flaws detection. In *Proceedings of 8th European Conference on Software Maintenance and Reengineering (CSMR'04)*, pages 223–232, Los Alamitos CA, 2004. IEEE Computer Society.

[Rie04] Matthias Rieger. Experiments on language independent duplication detection. Technical Report iam-04-002, University of Bern, Institute of Applied Mathematics and Computer Science, 2004.

[See06] Mauricio Seeberger. How developers drive software evolution. Master's thesis, University of Bern, January 2006.

[Tic01] Sander Tichelaar. *Modeling Object-Oriented Software for Reverse Engineering and Refactoring.* PhD thesis, University of Berne, December 2001.

2.3 Detailed Research Plan

The proposed research aims to ease the evolution of J2EE applications by conducting a systematic study to support their reverse engineering and quality assurance. We propose four closely related tracks to achieve this end:

1. construct a meta-model of J2EE and produce corresponding parsers,
2. develop metrics and quality detection strategies that take into account the specifics of J2EE,
3. develop visualizations that reveal the most important parts of J2EE applications.

We will outline the goals of each of these tracks and describe the steps we envisage to achieve these goals.

2.3.1 J2EE Meta-model

To analyze a software system, we first need to construct a model of it that conforms to an expected meta-model. Thus, a problem that we will tackle for the entire duration of the project is to build a unified meta-model of J2EE. The new meta-model will be based on FAMIX and it will extend it to express and analyze several issues: dependencies between Beans, dependencies between JSP Pages and Java source code, HTML anchors that point from one JSP to another, database structure, dependencies between Java source code and database structure.

We will also build parsers. While this is purely an engineering effort, we need parsers to be able to perform extensive analysis. However, we foresee that parsing the new sources of information will pose significant challenges.

In the case of JSP we need to parse the JSP files. We have already conducted an initial project, called j2moose[9], that transforms JSP into Java, and then uses the Eclipse infrastructure to produce the model. Currently, j2moose only parses the standard JSP code, and it ignores the custom tags that encapsulate the Java code. We plan to extend j2moose to take into account the taglib descriptor files that map the custom tags to Java classes.

In the case of EJB we need to parse the Java source code together with the configuration files. The goal is to recover the mapping between the Beans and the source code. For this we intend to explore two solutions: (1) extend the j2moose parser to also parse the XML files and export a complete model that includes JSP and EJB, or (2) build a separate tool to parse the XML and link the recovered information with the entities in the model produced by j2moose.

In the case of the databases, we will build tools to query the meta-data to recover the structure of the tables and then link these with the source code entities that access them.

We expect the meta-model to co-evolve with the produced parsers as more analyses will be constructed. We plan to start with extending the current solution for JSP. Soon afterwards, we will tackle EJB, and later on the connection to the database.

2.3.2 J2EE Metrics

J2EE is a relatively new technology, and no extensive research has been invested in quantifying the factors that influence the evolvability of J2EE applications. There exists, however, a large body of patterns and heuristics gathered by the engineering community from experience. That is why we will take as starting point the study and detection of existing patterns.

We first need a sample of J2EE applications to study. Besides analyzing open source projects that use J2EE as underlying technology (*e.g.* Liferay Portal[10], WordNet Web[11], MeshCMS[12]) we will also look for industrial applications.

The most straightforward metrics are those that measure the size of the different parts. Thus, we will start by building size metrics for the different parts. Here are some examples: the number

[9] http://j2moose.sourceforge.net/
[10] `http://sourceforge.net/projects/lportal/`
[11] `http://sourceforge.net/projects/wnwa/`
[12] `http://sourceforge.net/projects/meshcms/`

of Java source code lines in the JSP pages, the number of references to Java classes from JSP pages, the number of direct SQL queries both in JSP pages and in Java source code, the number of Beans.

An enterprise application has a complex domain model. In a well-designed J2EE application, the domain model will be at the core of the system and it will be formed by Java classes. Thus, the domain model represents the primary candidate for applying existing metrics-based detection techniques.

One hypothesis is that the domain model reflects possible flaws in the system, and that it should be the starting point for quality assurance analysis. On the one hand, we want to know how does a flaw in the domain model propagate in the rest of the system? For example, lacking an explicit abstraction can affect the database schema, which in turns affects the queries used to get the data. On the other hand, when dealing with a legacy database that cannot be easily changed, the domain model might be the one impacted by the design flaws in the database design.

The domain model should be isolated by a layer of Session Beans that is remotely invoked from the presentation layer. Furthermore, the access to the database should be encapsulated in another layer, potentially formed by Entity Beans. We intend to measure such interactions to detect the coupling between the different layers.

From a different perspective, when a large applications is developed by several developers and depends on several technologies, having a robust build process is crucial. That is why, we also want to investigate and to measure the build files to identify possible problems like code duplication. Furthermore, the analysis of the build files reveals how the application is intended to be deployed, information that can be used when measuring the dependencies between parts.

2.3.3 J2EE Visualization

Visualization is a useful reverse engineering tool, and can be used a first step in the process of designing automated detections. We plan to build visualizations along the entire period of the project: first to use visualization as an exploratory tool, and later to build a suite of visualizations that prove to be useful when analyzing J2EE applications.

First we want to visualize the separation of concerns by showing the components in the context of the layers and the relationships between them. With such a view we can possibly detect the dependencies that violate the layered architecture.

Another interesting piece of information is the accesses to the database from the Java classes and JSP pages. We expect to use such a visualization to detect whether the database access is localized or spread over the system.

The location in the file system of the various files is an important piece of information as it reveals the view of developers on the system. That is why we intend to construct a visualization that relates the source code to the file system to answer questions like: Are the JSP files separated from the Java classes?, Which JSP pages are grouped together? If there are several configuration files, are they placed together to the source code they configure? One possibility is to use a visualization similar to the Distribution Map.

We expect that one general challenge will be to find appropriate representations for displaying various languages. One possible solution we would like to experiment with is using VisualIDs. VisualIDs are automatically generated icons using an algorithm that preserves similarities based on a given input (*e.g.* names of entities). We would like to build classes of ids for each medium (*e.g.* JSP or EJB) and use them to display the complex relationships between the J2EE entities.

2.4 Timetable

Dr. Tudor Gîrba has recently started a PostDoc in our group, and we plan that Dr. Gîrba will occupy the PostDoc position in the context of this project. Mr. Stefan Reichhart finished his Masters thesis in April, and he will start his PhD on meta-modeling and measuring J2EE applications, if this project is accepted.

We expect to obtain the following results over the two years of the project. We expect a rough correspondence between bullet items below and publishable units.

Year 1	
Modeling J2EE	– extend FAMIX with a first version of the JSP meta-model – extend FAMIX with a first version of the EJB meta-model – extend FAMIX with a first version of the meta-model for representing databases – extend the current JSP parser to take into account custom tags – build a parser for EJB configuration files
J2EE metrics	– identify and parse case studies – develop size metrics – measure the quality of the domain models using object-oriented techniques
J2EE visualization	– design and implement first visualizations for showing the components and their relationships – design and implement a visualization for relating the source code to the file system – construct variations of the VisualIDs algorithm for representing the various languages
Year 2	
Modeling J2EE	– build an importer for recovering database structure – build a parser for SQL – extend the meta-model with relationships between Java classes and JSP pages to the database
J2EE metrics	– measure the coupling between layers – develop new quality detecting strategies
J2EE visualization	– refine the visualizations that show the components interaction – design and implement visualizations for showing how the database is accessed from the system
Year 3	
Modeling J2EE	– finalize the unified meta-model
J2EE metrics	– measure the quality of build files
J2EE visualization	– summarize a catalogue of visualizations useful for understanding J2EE applications

2.5 Significance of the Research

J2EE is the most widely-used technology in enterprise applications. Nevertheless, until now only little research has been spent on understanding the forces that influence the costs attached to evolving such applications. We strongly believe that analysis of J2EE applications, and more generally analysis of enterprise applications, will gain increasing importance in both research community and in industry.

Analyzing enterprise applications requires an extensive tool support. The availability of such a tool support is going to be a crucial factor in the success of research in this domain. We intend to capitalize on our Moose environment and FAMIX meta-model. Both Moose and FAMIX are already used in several universities (in Switzerland and in other European countries) for research in object-oriented reverse engineering and quality assurance.

We see the current project as an important step in creating a center of expertise in Switzerland in analyzing enterprise applications, and in consolidating the community that uses Moose by enabling new research paths. We intend to expand this community by holding anual workshops to enable the community to grow around Moose and FAMIX. This year we organized a workshop (FAMIX and Moose in Reengineering Research) colocated with TOOLS Europe 2007 in Zurich, where we provided researchers a forum to collaborate through ideas, data and implementation.

The research we propose addresses both fundamental and practical issues. The key venues for disseminating the results are the international, peer-reviewed conferences and journals in which we have consistently published our results.

In addition to publishing results in established venues, we also actively seek collaborations with industry, in particular to apply our analysis techniques to live and large case studies. We have recently completed a pilot project with Harman/BeckerAutomotive Systems (Germany) on assessing the quality of a large object-oriented system. We are initiating a pilot project with the Software Development Department of the Institut für Geistiges Eigentum (Switzerland) to study several large J2EE systems.