ICSE 2008 Tutorial on Pragmatic Design Quality Assessment

Tudor Gîrba
Inst. of Applied Mathematics
U. of Bern, Switzerland
girba@iam.unibe.ch

Michele Lanza
Faculty of Informatics
U. of Lugano, Switzerland
michele.lanza@unisi.ch

Radu Marinescu
Faculty of Computer Science
TU Timişoara, Romania
radum@cs.upt.ro

ABSTRACT

Quality control is paramount in every engineering discipline. Software engineering, however, is not considered a classical engineering activity for several reasons, such as intrinsic complexity and lack of rigor. In general, if a software system is delivering the expected functionality, only in few cases people see the need to analyze the internals.
In this tutorial we offer a pragmatic approach to analyzing the quality of software systems. On the one hand, we offer the theoretical background to detect quality problems by using and combining metrics, by analyzing the past through evolution analysis, and by providing visual evidence of the state of affairs in the system. On the other hand, as analyzing real systems requires adequate tool support, we offer an overview of the problems that occur in using such tools and provide a hands-on session with state-of-the-art tools used on a real case study.

1. PRESENTERS

Tudor Gîrba obtained his Ph.D. (*summa cum laude*) in November 2005 and is currently a postdoctoral researcher at the Software Composition Group, University of Berne, Switzerland. His interests lie in the area of software evolution, reverse engineering, reengineering, information visualization, quality assurance and meta-modeling. During his Ph.D. he developed the Hismo meta-model for evolution analysis. He is one of the key developers of the Moose reengineering environment and participated in the development of several other reverse engineering tools. He offers consulting services in the area of reengineering and quality assurance.

Contact:
Dr. Tudor Gîrba
Software Composition Group, Institut für Informatik
Universität Bern
Neubrueckstr. 10, 3012 Berne, Switzerland
web: http://www.iam.unibe.ch/~girba/
e-mail: girba@iam.unibe.ch

ICSE 2008 May 21-25 2008, Leipzig, Germany.

Michele Lanza is assistant professor at the faculty of informatics of the University of Lugano, where he leads a research group focusing on the areas of software evolution, visualization, and reverse engineering. He authored more than 40 technical papers and is author of the recently published book "Object-Oriented Metrics in Practice". He has mainly experience in building information visualization tools and approaches to deal with reverse engineering and evolution: His Ph.D. thesis (*summa cum laude*) received the *Ernst Denert Software Engineering Award* in 2003 for the thesis. He previously taught tutorials on "Software Evolution: Analysis & Visualization" together with Prof. Harald Gall at ICSE 2006 and OOPSLA 2007, with 25 and 15 participants, respectively.

Contact:
Prof. Michele Lanza
Faculty of Informatics
University of Lugano
Via G. Buffi 13, 6904 Lugano, Switzerland
tel. +41 58 666 4659 | fax +41 58 666 4536
web: http://www.inf.unisi.ch/lanza
e-mail: michele.lanza@unisi.ch

Radu Marinescu is an associate professor at the Politehnica University of Timişoara, where he leads the LOOSE research group focusing on the areas of object-oriented software evolution, quality assurance and reengineering. In 2002 he received his Ph.D. (*magna cum laude*) from the Politehnica University of Timişoara. He is author of the "Object-Oriented Metrics in Practice" book (Springer, 2006), and the recipient of an *IBM Eclipse Innovation Award* (2006). He has published over 25 papers related to metrics, quality assurance and software evolution and has served in the last years in more than 10 program committees of international conferences. He has been constantly working as a consultant and trainer for several well-known IT companies, such as Siemens, IBM, and Alcatel-Lucent.

Contact:
Prof. Radu Marinescu
LOOSE Research Group
Politehnica University Timişoara
Bvd. V.Parvan 2, 300223 Timişoara, Romania
web: http://www.cs.upt.ro/~radum/
e-mail: radum@cs.upt.ro

2. DURATION

1 full day

3. GOAL AND OBJECTIVES

The overall goal of this tutorial is to teach participants the good, the bad, and the ugly about the assessment of design quality, with a special emphasis on object-oriented design. In our research, we have explored a variety of metrics and visualization techniques to characterize, evaluate and even improve the design of large object-oriented systems and their evolution, in a language-independent manner. Specifically, the tutorial aims to answer the following questions:

- How can we use metrics to characterize, evaluate and improve the design of object-oriented systems?
- What are the usual pitfalls of the use of metrics, and how can we circumvent them?
- How can we, developers and designers, customize quality assessment techniques to make them applicable in their particular context?
- Once we have detected a flawed artifact in a system, how can we efficiently "cure" the system from that disease?
- Which are the key features to look for in a toolkit for quality assessment? And which are the useless ones?

We propose to tackle these questions by providing participants with:

- A detailed overview of state-of-the art techniques on software metrics, software evolution and software visualization
- A detailed survey of existing approaches and methodologies for quality assurance, reverse engineering and reengineering
- Practical knowledge and skills on how to use state-of-the-art tools to characterize, evaluate, and improve the design of object-oriented systems

4. SCOPE

Intended audience. This tutorial is open to all software practitioners, managers, students, and researchers who want to learn more about design quality assessment, software metrics, software evolution, software visualization and software reengineering.

Level. The level of the tutorial comprises both basic and advanced topics.

Required Prerequisites. The participants should have a basic background in software engineering and object-oriented programming, and be aware of simple software refactorings and metrics. However, the tutorial is designed in such a way that no specific previous knowledge is required.

5. TEACHING METHOD

The tutorial will be structured in two main parts:

1. **Theoretical Foundations.** This first part, given in the morning, consists of a set of lectures on a variety of correlated topics to provide the theoretical foundations to the tutorial attendees. The topics include metrics, evolution, software analysis, quality models, and visualization principles (see Section 7 for a detailed list.)
2. **Practical Skills.** This second part, given in the afternoon, introduces the tutorial attendees to state-of-the-art tools and will also give the attendees the possibility to experiment with them in a hands-on fashion. The goal of the hands-on session is for the participants to analyze the quality of an actual large object-oriented system and provide a short report of their findings to the other tutorial attendees.

The tutorial is closed by a discussion session where we summarize the tutorial by taking into account the findings of the attendees.

5.1 Technical Equipment

As far a technical equipment is concerned, we will need the following:

- A beamer to be connected to a laptop computer for the presentations and tool demos.
- (Wireless) Internet connection
- Participants should bring their own laptop to experiment with the tools.

Further, handouts of the presentation slides will be distributed (presumably in advance by the conference organization).

6. SUMMARY OF CONTENTS

Controlling quality is paramount in every engineering discipline. Software engineering, however, is not considered a classical engineering activity for several reasons, such as intrinsic complexity and lack of rigor, thus negatively impacting the consideration for internal quality. In general, if a software system is delivering the expected functionality, only few people -if any- care about the internals of the system.
Software metrics are widely used, specifically to measure and preserve software quality. However, defining, understanding and using them often looks like an overly complex activity, recommended only to "trained professionals".
In this tutorial, we demystify software metrics and show how they can be used to assess the size, quality and complexity of object-oriented software systems. Furthermore, we take into account the evolutionary aspect of software systems and we present both structural and evolutionary measurements in conjunction with visualization techniques.
In a practical setting, quality assurance requires adequate tools. That is why we go beyond theory and analyze the problems that arise in using such tools. The tutorial attendees will have the chance to exercise the lessons learned during the theoretical part of the tutorial, by using stat-of-the-art metrics and visualization tools in hands-on sessions.

The following sections provide an overview (including our own contributions) of the topics covered in the tutorial: metrics, evolution analysis, visualization and their usage in quality assessment [1].

6.1 Software Metrics

Metrics are a way to control quality [12]. In software engineering it is important and useful to measure systems, because otherwise we risk losing control because of their complexity. Losing control in such a case could make us ignore the fact that certain parts of the system grow abnormally or have a bad quality (*e.g.* code that is cryptic, uncommented, badly structured, or dead).

Software metrics are used to detect design problems as they quantify simple properties of design structures. Various software metrics have been defined to address the most important characteristics of good object-oriented design like complexity, cohesion, coupling and inheritance [5,8,29,40]. Lorenz and Kidd have worked on devising empirical threshold values which signify abnormal characteristics of design entities [41]. These thresholds were established based on the authors' experience with C++ and Smalltalk projects.

As there is currently an inflation of metrics, there is a serious need to have a rigorous approach for defining and using them. Briand *et al.* defined a unified framework for coupling measurement in object-oriented systems based on source model entities [6]. Based on these metrics they verified the coupling measurements at the file level using statistical methods and logical coupling information based on "ripple effects" [7]. Briand *et al.* described how coupling can be defined and measured based on dynamic analysis of systems [2]. This recent study shows that some dynamic coupling measures are significant indicators of change proneness and that they complement existing coupling measures that are based on static analysis.

In practice, appropriate tool support for metrics calculation is a must for performing quality assessments. Such tools are presented in [1, 55]. Unfortunately, many such tools do not go beyond the computation of a large number of metrics accompanied by the display of metrics in form of charts and by monitoring if some (oftentimes arbitrary) threshold values are not exceeded by the software system under scrutiny.

We emphasize that a metric alone cannot serve the aforementioned goal of controlling design quality. Metrics taken in isolation cannot help to answer all the questions about a system. The *bottom-up* approach, *i.e.* going from abnormal numbers to the recognition of design flaws is impracticable because the symptoms captured by single metrics, even if perfectly interpreted, may occur in several flaws: the interpretation of individual metrics is too *fine grained* to indicate a particular design problem. This tutorial reveals how metrics can be combined in order to serve the identification and location of design problems and thus contribute to controlling design quality.

To support the detection and location of design problems in a system, we introduced a technique – called *detection strategy* [36,43] – for formulating metrics-based rules that capture deviations from good design principles and heuristics like those defined by Riel [51] or Martin [44]. Using detection strategies an engineer can directly localize classes or methods affected by a particular design flaw (*e.g.* God Class), rather than having to infer the real design problem from a large set of abnormal metric values. We have defined such detection strategies for capturing more than ten important flaws of object-oriented design and we have implemented them in both iPlasma and Moose reengineering environments [49].

[1]Please note that we use "we" as a generic term to refer to either one of the authors to simplify the text

6.2 Software Evolution

The importance of *modeling and analyzing* software evolution was pioneered in the early 1970's with the work of Lehman and Belady who established that as systems evolve, they become more complex, and consequently more resources are needed to preserve and simplify their structure [37]. Yet, it was only in recent years that extensive research has been carried out on exploiting the wealth of information residing in versioning repositories for purposes like reverse engineering or cost estimation. Problems like software aging [50] and code decay [15] gained increasing recognition both in academia and in industry.

There are several approaches that analyze the influence of changes in an evolving software system. Lehmann *et al.* explored the implication of evolution on quality and software maintenance in general by analyzing the differences between consecutive versions using simple measurements [38, 39].

Gold and Mohan defined a framework to understand the conceptual changes in an evolving system [28]. Based on measuring the detected concepts, they could differentiate between different maintenance activities. Mockus and Votta classified the changes [46] into corrective, adaptive, inspection, perfective, and other types changes, by analyzing the descriptions in the change logs. Mockus and Weiss used history measurements for developing a method for predicting the risk of software changes [47]. Some examples of such measurements are: the number of modules touched, the number of developers involved, or the number of changes. Extracting architectural properties from large open source systems such as the Mozilla system has been addressed by Godfrey *et al.* [27]. Their work relied on PBS [18] which is a reverse engineering workbench containing the Relational Algebra tool Grok. PBS does not consider the visualization of metrics to characterize abstracted entities and relationships, or to filter out the information of minor interest leading to more condensed and comprehensible views.

Gall *et al.* pioneered the research on detecting logical coupling between parts of the system that change in the same time [20]. Zimmermann *et al.* focused on a mechanism to warn developers that: "Programmers who changed function x also changed ..." [60]. Further, Ying *et al.* applied data mining techniques to the change history of the code base to identify change patterns to recommend potentially relevant source code for a particular modification task [59].

All these individual techniques provide facets for assessing the quality of software systems. However, we established that to understand software evolution as a whole, we need a means to combine and compare these different analyses [21]. Thus, we created the Hismo meta-model to support the expression of these analyses and we used it to develop various evolution analyses on top of the Moose reengineering environment [22]. One example of such an analysis is Yesterday's Weather, an algorithm to check whether the entities that were changed recently will be among those changed in the near future [24]. We have also used concept analysis to identify patterns of co-change [23].

6.3 Software Visualization

The goal of information visualization is to *visualize any kind of data* [54]. Applications of information visualization are so frequent and common, that most people do not notice them: examples include metereology (weather maps), geography (street maps), transportation (train tables and metro maps), etc.

In the more specific field of software engineering, UML is probably the most used visual notation [19], but many more visualization techniques have been proposed dealing with various aspects of software analysis.

One of the most spread metaphor used is to map the structure of the code on a graph and represent it accordingly. Thus, tools like such as Rigi [48] and SHriMP [52] were built.

We have also been involved in building such tools, most notably CodeCrawler [32] and Mondrian [45]. A technique that we coined is the *polymetric view* [35]. The polymetric view enriches the graph visualization with metrics mapped on the size and color of the nodes and edges. Class Blueprint is one example of a polymetric view that shows the internals of a class in terms of methods, attributes and their inter-relations [13]. Other examples consist in applying polymetric views to display the evolution of classes [34], or to display dynamic information [14].

Visualization has proven to be a key technique for software evolution analysis, mainly due to the huge amounts of information that need to be processed and understood. One of the first application of visualization in evolution analysis is embodied in Seesoft, a tool for visualizing line oriented software statistics such as the age or stability of a line of code [17]. Jazayeri *et al.* analyzed the stability of the architecture [30,31] by using colors to depict the changes over a period of releases. Similarly Wu *et al.* describe an Evolution Spectrograph [57] that visualizes a historical sequence of software releases. Rysselberghe and Demeyer used a simple visualization based on information in version control systems to provide an overview of the evolution of systems [53].

We believe that classes are best understood in their context like their inheritance hierarchies, however only few efforts have been invested into understanding the evolution of hierarchies. Collberg *et al.* used graph-based visualizations to display which parts of class hierarchies were changed [9]. They provide a color scale to distinguish between newer and older changes. We have developed a polymetric view that shows the evolution of classes at system level emphasizing new and old as well as changed and or stable classes [26].

Changes are performed by developers. Different approaches have been developed to analyze author information available from the versioning system. Ball and Eick [4] have represented lines of code as lines and mapped colours to represent the authors. Xiaomin Wu *et al.* visualized the change log information to provide an overview of the active places in the system as well as of the authors activity [58]. Eick *et al.* proposed several visualizations to show how developers modify the system using colors and third dimension [16]. We have developed visualizations to show how which developers own which parts of the code [25] and which developers copy from each other [3].

Visualization has become a key technique for analyzing large and complex data sets. We have gained extensive knowledge of this domain, by building a number of visualization tools [10, 11, 32, 45, 56], and have come to the conclusion that they are also a key factor in understanding software quality.

6.4 Summary

The three tutorial speakers have performed research in quality assurance, software reverse engineering, evolution, visualization, and reengineering since 10 years. The tutorial strongly builds on our PhD theses [21, 33, 42], and on the papers we have published. Furthermore, the tutorial builds on our recently published book *Object-Oriented Metrics in Practice* [36].

Our claim is that quality needs to be regarded from various perspectives and in this tutorial we show how metrics, visualization and evolution analyses can be combined to help quality assessment. Furthermore, based on our extensive experience in building reverse engineering tools we distill the lessons learnt related to using and building reengineering tools, and we provide a hands-on session for attendees to practice these lessons.

7. STRUCTURE OF CONTENTS

The tutorial is structured in two main part, organized as follows:

1. Theoretical Foundations
   (a) Introduction
      - What is software quality?
      - An overview of the field and introduction of key concepts

   (b) Software Metrics
      - Myths and truths about metrics.
      - Software quality models (the ISO 9126 model).
      - Detection Strategies.
      - Design Disharmonies: Identity Disharmonies, Classification Disharmonies, Collaboration Disharmonies.

   (c) Software Visualization
      - What is software visualization?
      - Overview of the field
      - Visualization and metrics: Polymetric Views, Class Blueprints.
      - Recent trends in visualization.

   (d) Software Evolution
      - What is software evolution?
      - What can we learn from the past?
      - Overview of the field
      - A quick introduction to the field of software refactorings.
      - Eradicating disharmonies through refactorings.

2. Practical Skills
   (a) Tools for Metrics, Visualization
      - What to look for in a metrics tool: navigation, code inspection, causality of the numbers, correlation of numbers.
      - Brief survey of the available tools
      - Tool support: the Moose reengineering environment, the iPlasma environment

(b) Hands-on: step-by-step using Moose and iPlasma on a sample case study.

(c) Hands-on: producing a report of problems found in a case study.

(d) Wrap-up

- Discussion of lessons learned regarding metrics, visualization and evolution.
- Discussion of possible refactorings.
- Discussion on the tool support: what is good, what else would be best to have.

Acknowledgments: Gîrba gratefully acknowledge the financial support of the Hasler Foundation for the project "Enabling the evolution of J2EE applications through reverse engineering and quality assurance" (Project no. 2234, Oct. 2007 - Sept. 2010).

8. REFERENCES

[1] J. Alghamdi, R. Rufai, and S. Khan. OOMeter: A Software Quality Assurance Tool. In *ICSM*, 2005.

[2] E. Arisholm, L. C. Briand, and A. Føyen. Dynamic Coupling Measurement for Object-Oriented Software. *Transactions on Software Engineering*, 30(8):491–506, 2004.

[3] M. Balint, T. Gîrba, and R. Marinescu. How developers copy. In *Proceedings of International Conference on Program Comprehension (ICPC 2006)*, pages 56–65, 2006.

[4] T. Ball and S. Eick. Software visualization in the large. *IEEE Computer*, 29(4):33–43, 1996.

[5] J. Bieman and B. Kang. Cohesion and reuse in an object-oriented system. In *Proceedings ACM Symposium on Software Reusability*, Apr. 1995.

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

[7] L. C. Briand, J. W. Daly, and J. K. Wüst. Using coupling measurement for impact analysis in object-oriented systems. In *Proceedings of the 21st International Conference on Software Engineering (ICSE 1999)*, pages 475–482, 1999.

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

[9] C. Collberg, S. Kobourov, J. Nagra, J. Pitts, and K. Wampler. A system for graph-based visualization of the evolution of software. In *Proceedings of the 2003 ACM Symposium on Software Visualization*, pages 77–86, New York NY, 2003. ACM Press.

[10] M. D'Ambros and M. Lanza. Reverse engineering with logical coupling. In *Proceedings of WCRE 2006 (13th Working Conference on Reverse Engineering)*, pages 189 – 198, 2006.

[11] M. D'Ambros and M. Lanza. Software bugs and evolution: A visual approach to uncover their relationship. In *Proceedings of CSMR 2006 (10th IEEE European Conference on Software Maintenance and Reengineering)*, pages 227 – 236. IEEE Computer Society Press, 2006.

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

[13] S. Ducasse and M. Lanza. The class blueprint: Visually supporting the understanding of classes. *Transactions on Software Engineering (TSE)*, 31(1):75–90, Jan. 2005.

[14] S. Ducasse, M. Lanza, and R. Bertuli. High-level polymetric views of condensed run-time information. In *Proceedings of 8th European Conference on Software Maintenance and Reengineering (CSMR'04)*, pages 309–318, Los Alamitos CA, 2004. IEEE Computer Society Press.

[15] S. Eick, T. Graves, A. Karr, J. Marron, and A. Mockus. Does code decay? assessing the evidence from change management data. *IEEE Transactions on Software Engineering*, 27(1):1–12, 2001.

[16] S. Eick, T. Graves, A. Karr, A. Mockus, and P. Schuster. Visualizing software changes. *IEEE Transactions on Software Engineering*, 28(4):396–412, 2002.

[17] S. G. Eick, J. L. Steffen, and S. Eric E., Jr. SeeSoft—a tool for visualizing line oriented software statistics. *IEEE Transactions on Software Engineering*, 18(11):957–968, Nov. 1992.

[18] P. Finnigan, R. Holt, I. Kalas, S. Kerr, K. Kontogiannis, H. Müller, J. Mylopoulos, S. Perelgut, M. Stanley, and K. Wong. The software bookshelf. *IBM Systems Journal*, 36(4):564–593, November 1997.

[19] M. Fowler. *UML Distilled*. Addison Wesley, 2003.

[20] H. Gall, K. Hajek, and M. Jazayeri. Detection of logical coupling based on product release history. In *Proceedings International Conference on Software Maintenance (ICSM '98)*, pages 190–198, Los Alamitos CA, 1998. IEEE Computer Society Press.

[21] T. Gîrba. *Modeling History to Understand Software Evolution*. PhD thesis, University of Berne, Berne, Nov. 2005.

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

[23] T. Gîrba, S. Ducasse, A. Kuhn, R. Marinescu, and D. Raţiu. Using concept analysis to detect co-change patterns. In *Proceedings of International Workshop on Principles of Software Evolution (IWPSE 2007)*, 2007. To appear.

[24] T. Gîrba, S. Ducasse, and M. 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, Sept. 2004. IEEE Computer Society.

[25] T. Gîrba, A. Kuhn, M. Seeberger, and S. 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.

[26] T. Gîrba, M. Lanza, and S. 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.

[27] M. Godfrey and E. H. S. Lee. Secrets from the Monster: Extracting Mozilla's Software Architecture. In *Proceedings of Second Symposium on Constructing Software Engineering Tools (CoSET'00)*, June 2000.

[28] N. Gold and A. Mohan. A framework for understanding conceptual changes in evolving source code. In *Proceedings of International Conference on Software Maintenance 2003 (ICSM 2003)*, pages 432–439,

Sept. 2003.

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

[30] M. Jazayeri. On architectural stability and evolution. In *Reliable Software Technologies-Ada-Europe 2002*, pages 13–23, Berlin, 2002. Springer Verlag.

[31] M. Jazayeri, H. Gall, and C. Riva. Visualizing Software Release Histories: The Use of Color and Third Dimension. In *Proceedings of ICSM '99 (International Conference on Software Maintenance)*, pages 99–108. IEEE Computer Society Press, 1999.

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

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

[34] M. Lanza and S. 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.

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

[36] M. Lanza and R. Marinescu. *Object-Oriented Metrics in Practice*. Springer-Verlag, 2006.

[37] M. Lehman and L. Belady. *Program Evolution: Processes of Software Change*. London Academic Press, London, 1985.

[38] M. Lehman, D. Perry, and J. Ramil. Implications of evolution metrics on software maintenance. In *Proceedings IEEE International Conference on Software Maintenance (ICSM'98)*, pages 208–217, Los Alamitos CA, 1998. IEEE Computer Society Press.

[39] M. Lehman, D. Perry, J. Ramil, W. Turski, and P. Wernick. Metrics and laws of software evolution–the nineties view. In *Proceedings IEEE International Software Metrics Symposium (METRICS'97)*, pages 20–32, Los Alamitos CA, 1997. IEEE Computer Society Press.

[40] W. Li and S. Henry. Maintenance metrics for the object oriented paradigm. *Proceedings of the First International Software Metrics Symposium.*, pages 52–60, May 1993.

[41] M. Lorenz and J. Kidd. *Object-Oriented Software Metrics: A Practical Guide*. Prentice-Hall, 1994.

[42] R. Marinescu. *Measurement and Quality in Object-Oriented Design*. PhD thesis, Department of Computer Science, Politehnica University of Timişoara, 2002.

[43] R. 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.

[44] R. C. Martin. *Agile Software Development. Principles, Patterns, and Practices*. Prentice-Hall, 2002.

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

[46] A. Mockus and L. Votta. Identifying reasons for software change using historic databases. In *Proceedings of the International Conference on Software Maintenance (ICSM 2000)*, pages 120–130. IEEE Computer Society Press, 2000.

[47] A. Mockus and D. Weiss. Predicting risk of software changes. *Bell Labs Technical Journal*, 5(2), Apr. 2000.

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

[49] O. Nierstrasz, S. Ducasse, and T. 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.

[50] D. L. Parnas. Software aging. In *Proceedings 16th International Conference on Software Engineering (ICSE '94)*, pages 279–287, Los Alamitos CA, 1994. IEEE Computer Society.

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

[52] M.-A. D. Storey and H. 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.

[53] F. Van Rysselberghe and S. Demeyer. Studying software evolution information by visualizing the change history. In *Proceedings 20th IEEE International Conference on Software Maintenance (ICSM '04)*, pages 328–337, Los Alamitos CA, Sept. 2004. IEEE Computer Society Press.

[54] C. Ware. *Information Visualization*. Morgan Kaufmann, 2000.

[55] S. Website. Sdmetrics, 2005.

[56] R. Wettel and M. Lanza. Program comprehension through software habitability. In *Proceedings of ICPC 2007 (15th International Conference on Program Comprehension)*, pages 231–240. IEEE CS Press, 2007.

[57] J. Wu, R. Holt, and A. Hassan. Exploring software evolution using spectrographs. In *Proceedings of 11th Working Conference on Reverse Engineering (WCRE 2004)*, pages 80–89, Los Alamitos CA, Nov. 2004. IEEE Computer Society Press.

[58] X. Wu, A. Murray, M.-A. Storey, and R. Lintern. A reverse engineering approach to support software maintenance: Version control knowledge extraction. In *Proceedings of 11th Working Conference on Reverse Engineering (WCRE 2004)*, pages 90–99, Los Alamitos CA, Nov. 2004. IEEE Computer Society Press.

[59] A. T. Ying, G. C. Murphy, R. Ng, and M. C. Chu-Carroll. Predicting Source Code Changes by Mining Change History. *IEEE Transactions on Software Engineering*, 30(9):574–586, 2004.

[60] T. Zimmermann, P. Weißgerber, S. Diehl, and A. Zeller. Mining version histories to guide software changes. In *26th International Conference on Software Engineering (ICSE 2004)*, pages 563–572, Los Alamitos CA, 2004. IEEE Computer Society Press.