Software Quality Assessment of Open Source Software

Georgios Gousios, Vassilios Karakoidas, Konstantinos Stroggylos, Panagiotis Louridas, Vasileios Vlachos and Diomidis Spinellis

Athens University of Economics and Business, Patission 76, Athens, Greece

Abstract

The open source software ecosystem comprises more than a hundred thousand applications of varying quality. Individuals and organizations wishing to use open source software packages have scarce objective data to evaluate their quality. However, open source development projects by definition allow anybody to read, and therefore evaluate their source code. In addition, most projects also publish process-related artefacts, such as bug databases, mailing lists, and configuration management system logs. The software quality observatory is a platform that uses these product and process data sources to automatically evaluate the quality of open source projects. A plugin-based service-oriented architecture allows the mixing and matching of metrics extraction suites, source code repositories, and transformation filters. The resulting platform is aimed at IT consultants and managers, the open source community, and researchers.

Keywords: Open Source, Software Quality

1. Introduction

A large number of open source software (OSS applications exist in the free software ecosystem; on last count, the Sourceforge OSS hosting web site reports 138,000 projects and almost 1,5 million registered developers. As the OSS makes significant inroads into the commercial market sector, its *quality* is questioned and often becomes an issue of controversy (Microsoft 2006; Enterprise Management Associates 2006). The large number of OSS packages offering equivalent or overlapping functionality combined with the current lack of objective evaluation criteria makes the choice of a specific package difficult.

A well-known conjecture in software engineering is that external quality characteristics are correlated to internal quality characteristics and thus source code metrics provide useful data for the assessment of its quality. Uniquely, open source software allows us to examine the actual code and perform white box testing and analysis of it (Spinellis 2006). In most open source projects we can also access their version control system, mailing lists and bug management databases. In this paper, we present an overview of the Software Quality Observatory for Open Source Software

(SQO-OSS), a framework for the automatic evaluation of source code. The SQO-OSS platform aims to combine well-known process and product metrics, with novel quality indicators extracted from currently under-utilised data sources.

The remainder of this paper is organized as follows; in Section 2, we discuss in brief the work carried out in the area of automated software quality assessment. In Section 3, we describe the architecture of the SQO-OSS platform core; we also present a preliminary version of the platform extension system. In Section 4 we present a list of the expected results, while Section 5 concludes this paper.

2. Software Quality

Software quality is an abstract concept that is perceived and interpreted differently based on one's personal views and interests. To dissolve this ambiguity, ISO/IEC-9126 (International Organization for Standardization 2001) provides a framework for the evaluation of software quality. It defines six software quality attributes, often referred to as *quality characteristics*:

- **Functionality:** Whether the software performs the required functions
- **Reliability:** Refers to maturity, fault tolerance and recoverability
- **Usability:** Refers to the effort required to understand, learn, and operate the software system
- **Efficiency:** Refers to performance and resource use behaviour
- **Maintainability:** Refers to the effort required to modify the software
- **Portability:** Refers to the effort required to transfer the software to another environment

The last five characteristics are not related to the task performed by the software and therefore are regarded as non-functional attributes. In many cases though software requirements and testing methodologies are mostly focused on functionality and pay little if any attention to non-functional requirements. Since NFRs affect the perceived quality of software (quality in use), failure to meet them often leads to late changes and increased costs in the development process. On the other hand, there are several challenges and difficulties, in assessing non-functional quality characteristics for software products in general, since the evaluation of quality in use is partly affected by users' knowledge and experience. For example, security is a non-functional requirement that needs to be addressed in every software project. Therefore badly-written software may be functional, but subject to buffer overflow attacks.

2.1 Related Work

Popular metrics suites are (among others) Order of Growth (usually referred to as Big-O-notation) (Cormen et al. 2001), Halstead's Complexity Measures (Halstead 1977) and McCabe's Cyclomatic Complexity (McCabe 1976; McCabe et al. 1989; McCabe et al. 1994). In the context of object-oriented systems the metrics used more commonly include the suites proposed by Henry and Kafura (Henry et al. 1981; Henry et al. 1984), Chidamber and Kemerer (Chidamber et al. 1991; Chidamber et al. 1994), Li and Henry (Li et al. 1993), Lorenz and Kidd (Lorenz et al. 1994) and Briand (Briand et al 1997; Briand et al. 1999). A comparative evaluation of the metrics presented above and also others in various development contexts is presented in (Xenos et al. 2000).

Software metrics to some extent can reflect software quality, and thus are widely used in software quality evaluation methods (Boehm et al. 1976) and models (Bansiya et al. 2002). Several studies attempt to correlate software metrics with quality (Subramanyam et al. 2003; Kan 2002). Basili (Basili et al. 1996) show that 5 out of 6 of the object oriented metrics proposed by Chidamber and Kemerer are useful in predicting class fault-proneness from the early development phases. Li and Henry (Li et al. 1993) analyzed object oriented systems trying to show that there is a relation between metrics and the maintenance effort required to keep a system up to date with changing requirements. Their study indicates that a combination of metrics can be used to predict the maintainability of an object oriented system, which is an indicator of its quality. Other recent studies attempt to validate the significance of the various metrics proposed in the literature (Briand et al. 2000).

Another influential factor for software quality is its design. Adhering to proven design principles, such as sub-system decoupling, allows a software system to evolve effortlessly as it makes modifications cheaper. Object-oriented systems expose this behavior more than ones written in procedural languages, because the software engineer can employ many powerful mechanisms such as inheritance, polymorphism, encapsulation and established design patterns. Therefore, by evaluating the quality of the design of a system one can estimate its overall quality.

A number of studies attempt to correlate attributes of the design of a system to its quality, measured by defect density and maintenance effort, and provide predictive models based on the values of the metrics for these attributes (Abreu et al. 1995; Abreu et al. 1996). The development effort and its relation to the cost of software as influenced by the duration of its development has also been examined (Angelis et al. 2005). Others, define formal models for object oriented design and use them to define object-oriented metrics, such as the ones proposed by Chidamber and Kemerer, in order to automate the evaluation of the design of a system (Chatzigeorgiou 2003; Reissing 2001).

2.2 Automated quality assessment tools

A number of researchers have attempted to take advantage of the rapid growth of the availability of source code and process data in OSS projects in order to design new techniques for estimating a project's quality. Many of those studies use data mining and other techniques in an attempt to associate source code changes and metrics extracted from source code repositories with software defects (Williams et al. 2005; Purushotaman et al. 2005; Li et al. 2006), software quality (Sarkar et al. 2007) or maintainability effort (Zimmermann et al. 2005; Tsantalis et al. 2005). The availability of various information sources such as commit logs and bug management databases has also led to the creation of a number of projects that attempt to either create large collections of product and process metrics for numerous OSS projects (Litvinenko et al. 2007; Howison et al. 2006) or extract, aggregate and correlate the information in these sources to allow for easier further study (Cubranic et al. 2005; Johnson et al. 2005; Bevan et al. 2005).

3. The SQO-OSS Design

The design of the SQO-OSS platform is based on three non-functional properties we deemed important. These are:

- **Simplicity:** Existing tools should be integrated in the system easily without the understanding of complex APIs and XML Schema. We will use in Section 3.1 *wc* as a litmus test; it should be incorporated in SQO-OSS readily.
- **Extensibility:** The tools and the specifications we will develop will be open.
- **Interoperability:** The goal here is to create a platform that can encapsulate other applications. We will use XML and web services where appropriate to enhance interfacing with them.

The implementation of SQO-OSS will be based on a plugin architecture, comprising of components that communicate through a common data store. Everything that can be a plugin will be a plugin, which means that we will have (see Figure 1).

- Plugins for metrics calculation, data processing, and results processing
- A plugin manager for insertions, removals, and plugins updates
- A database for keeping track of plugins, measurements, and raw data

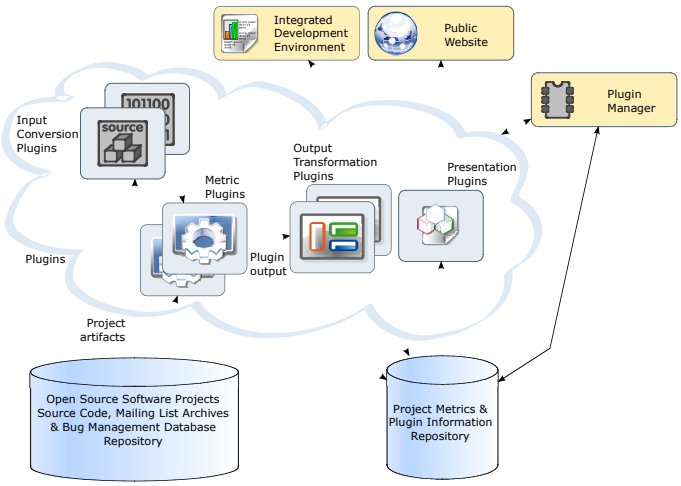

Figure 1: *The SQO-OSS platform building blocks*

When a new data source (for instance, a new major release of the Linux kernel) becomes available, the data will be copied on the SQO-OSS servers and the system database will be updated accordingly. All references to the data in SQO-OSS will be realised through appropriate queries to the database. Similarly, when a new plugin (or a new version of a plugin) becomes available, the system database will again be updated accordingly. The plugin manager will carry out the update process.

Having both a data source and appropriate plugins, we will be able to run the plugin on the data source, provided that the plugin is suitable for the particular data source. In particular, the *metrics plugins* will be of the following kinds:

- **Source code plugins** that calculate a metric directly on source code (e.g., count the number of lines of code)
- **Data mining plugins** that calculate a metric using structured and semi-structured information from various parts of a project's process data set using data mining techniques
- **Statistical analysis plugins** that use structured and semi-structured information from various parts of a project's process data set and the results of other plugins in order to build statistical estimation models. Those models will predict certain events in the project development cycle.

Some of the metrics plugins may need the data in a particular format. The necessary conversion will be the responsibility of *input conversion plugins*. For instance, several metrics plugins may calculate measurements based on a project's configuration management system logs. Instead of replicating the functionality of reading and

parsing the logs of configuration management tools like CVS and SVN, the metrics plugins will use the facilities offered by an appropriate CVS-read or SVN-read plugin whose responsibility will be to render the data in the required format.

After their execution, the metrics plugins will output their results to the system database. Some plugins may do that directly, but for some others their output will need to be converted to appropriate format to be entered in the database, in which case the metrics plugin output will be chained to suitable *output transformation plugins*. To get the results out of the SQO-OSS database we will need *presentation plugins*. These can be a simple matter of reading data from the database and returning them as text, or they may be converting to HTML or some other required format. These plugins will be capable of chaining, so that the output of one of them can be the input of a next one in a transformation sequence.

3.1 Examples

Suppose we want to add a new plugin that will allow us to count the number of lines of source code. This plugin will be in fact the UNIX *wc* utility; in effect we will be teaching SQO-OSS about it using the following message:

```
<plugin>
  <name>wc</name>
  <version>1</version>
  <applyto>allsrc</applyto>
  <provides>
    <metric>LOC</metric>
  </provides>
  <cmd>xargs cat | wc -l</cmd>
</plugin>
```

A slightly more involved example, involving the use of the Chidamber and Kemerer (Chidamber et al. 1994) object oriented metrics suite for Java (Spinellis 2005), would be:

```
<plugin>
  <name>ckjm</name>
  <version>5</version>
  <applyto>javaclasses</applyto>
  <provides>
      <id>classname</classname>
      <metric>WMC</metric> <metric>DIT</metric>
      <metric>NOC</metric> <metric>CBO</metric>
      <metric>RFC</metric>
```

```
  </provides>
  <cmd>java -jar \$JAVALIB/ckjm-1.5.jar</cmd>
</plugin>
```

In order to call the *wc* plugin on a specific target (release 7 of the FreeBSD operating system) we would send the following message to SQO-OSS:

```
<plugin>
  <name>wc</name>
  <version>1</version>
  <target>
    <projectid>FreeBSD</projectid>
    <version>RELENG_7</version>
  </target>
</plugin>
```

In this message, the \texttt{target} refers to a combination of the project ID and its version by which a system identifies a given dataset. The results will be saved as text in the system database.

As explained above, all measurements will be kept into the system database, so we need a way to retrieve them. To get the output from the *wc* call we would use:

```
<transformer>
  <name>text</name>
  <source>measurementid</source>
</transformer>
```

We will be able to chain the output to obtain the desired result:

```
<transformer>
  <name>HTML</name>
  <source>
    <transformer>
      <name>text</name>
      <source>measurementid</source>
    </transformer>
  </source>
</transformer>
```

Similarly, a metric plugin call and output can be combined into a single call in the obvious way.

The basic interface to the system is messages, like the ones we outlined above. This allows us to have the same capabilities offered both through a command line interface

and services calls. The plugin descriptors show that plugins are independent from each other. Hence, plugins for data access are independent from metrics plugins, so that they can mix and match as appropriate. Also, we cannot assume that all plugins will be written in the same language, or that, indeed, we will write all the plugins. The system will accept any plugin that runs on its operating system platform, as long as it comes with an acceptable descriptor.

4. Expected Results

The software quality observatory for open source software forms a holistic approach to software quality assessment, initially targeted towards the specific requirements set by its OSS underpinnings. The three main contributions of the SQO-OSS project are:

- An extensible platform for software quality assessment that can either be used stand-alone or be incorporated into the development process through extensions to popular development tools. The quality assessment plug-ins are fully independent from the main platform. The platform will thus support all programming languages for which there exist quality metric tools.
- Combination of readily available metrics with novel techniques involving mining software repositories for project evolution assessment and correlating mailing list information to bug management database entries. Our main scientific contribution can be summarised to the correlation of a large number of product and process metrics with time in an effort to predict future project behaviour.
- An observatory of quality assessed open source applications. The observatory is going to be updated in an automatic fashion to include the latest versions of the assessed software. The user will also be allowed to customise the observatory results to his/her own needs by means of predefined or custom made quality models

The SQO-OSS project results will be of interest to a wide number of audiences. The main target of our focus is on the following categories of users:

- **IT consultants and IT managers:** Those users need to make informed decisions on software to include into business processes. They are mainly interested in software combining functionality with a proven record and a predictable future. The software quality observatory for open source software will enable them to choose among already evaluated software packages and also provide them with the capability to evaluate them in detail according to their needs, as they will have the data and the ability to quantify the quality of a software project.
- **OSS community:** Our aim is to keep open two-way communication channels with the OSS community. OSS developers need free tools to assist them

with the tedious task of quality assurance. While such tools already exist (Johnson et al. 2005; Litvinenko et al. 2007) they are not open to the community to take advantage of. The SQO-OSS platform will enable OSS developers to introduce formal quality control to their projects, which will in turn lead to the improved of the community generated software. Furthermore, SQO-OSS will promote the use of OSS by providing scientific evidence of its perceived quality.

- **Researchers:** The SQO-OSS framework can serve as a common testbed for the development of quality evaluation techniques. It will also allow for quick prototyping of software quality models by enabling researchers to combine metric results in arbitrary ways.

5. Conclusions

The SQO-OSS system is a platform modelled around a pluggable, extensible architecture that enables it to incorporate various types of data sources and be accessible through different user interfaces.

Over the past years a new market has been created based on products and value added services built on top of FLOSS. In order to allow it to grow, its products must somehow be comparable to each other, and thus they need to undergo some kind of standardization. This is the purpose of a number of research projects currently funded by the EU, besides SQO-OSS. FLOSSMetrics (Gonzales-Barahona 2005) stands for Free/Libre/Open Source Software Metrics and Benchmarking. Its main objective is to construct, publish and analyse a large scale database with information and metrics about Libre software development coming from several thousands of software projects, using existing methodologies and tools already developed. QUALOSS stands for QUALity of Open Source Software. Its main objective is to provide quality models that will allow European SMEs involved in software development to select FLOSS components of high quality and integrate them in their systems and products. QualiPSo (Quality Platform for Open Source Software) aims to prepare the industry for OSS and vice versa.

SQO-OSS aims to correlate OSS data from various information sources with the quality characteristics described in Section 2.1. It also intends to cooperate with the other related projects, in an attempt to allow all the involved parties to obtain results faster and combine their efforts in order to leverage the growth and adoption of OSS.

Currently, we are developing prototypes to test the design we presented here. Our initial findings strongly suggest us to maintain our loosely-coupled component approach across all the services that will be designed for the system. Therefore, we plan to base our system on a software bus architecture that will automatically handle

the management of plug-ins and then implement all quality assessment plug-ins as individual programs with custom wrappers, as required by the software bus we will select to base SQO-OSS on.

Acknowledgement: This work was funded by the European Community's Sixth Framework Programme under the contract IST-2005-033331 ``Software Quality Observatory for Open Source Software (SQO-OSS).

6. References

Lefteris Angelis and Panagiotis Sentas (2005). Duration analysis of software projects. In *Proceedings of the 10th Panhellenic Conference on Informatics* (PCI '05), Volos, Greece.

Rajendra K. Bandi, Vijay K. Vaishnavi, and Daniel E. Turk (2003). *Predicting maintenance performance using object-oriented design complexity metrics*. IEEE Transactions on Software Engineering, 29(1):77–87.

Jagdish Bansiya and Carl G. Davis (2002). *A hierarchical model for object-oriented design quality assessment*. IEEE Transactions on Software Engineering, 28(1):4–17. (doi:10.1109/32.979986)

Victor R. Basili, Lionel C. Briand, and Walcelio L. Melo (1996). *A validation of object-oriented design metrics as quality indicators*. IEEE Transactions on Software Engineering, 22(10):751–761.

Jennifer Bevan, Jr. E. James Whitehead, Sunghun Kim, and Michael Godfrey (2005). Facilitating software evolution research with kenyon. In *ESEC/FSE-13: Proceedings of the 10th European software engineering conference* held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering, pages 177–186. (doi:10.1145/1081706.1081736)

B. W. Boehm, J. R. Brown, and M. Lipow (1976). Quantitative evaluation of software quality. In *ICSE '76: Proceedings of the 2nd International Conference on Software Engineering*, pages 592–605, Los Alamitos, CA, USA. IEEE Computer Society Press.

Lionel Briand, Prem Devanbu, and Walcelio Melo (1997). An investigation into coupling measures for C++. In *ICSE '97: Proceedings of the 19th International Conference on Software Engineering*, pages 412–421, New York, NY, USA. ACM Press.

Lionel C. Briand, Sandro Morasca, and Victor R. Basili (1999). *Defining and validating measures for object-based high-level design*. IEEE Transactions on Software Engineering, 25(5):722–743. (doi:10.1109/32.815329)

Lionel C. Briand, Jürgen Wüst, John W. Daly, and D. Victor Porter (2000). *Exploring the relationships between design measures and software quality in object-oriented systems*. Journal of Systems and Software, 51(3):245–273.

Alexander Chatzigeorgiou (2003). *Mathematical assessment of object-oriented design quality*. IEEE Transactions on Software Engineering, 29(11):1050–1053. (doi:10.1109/TSE.2003.1245306)

Shyam R. Chidamber and Chris F. Kemerer (1991). Towards a metrics suite for object oriented design. In OOPSLA '91: Proceedings of the 6th annual ACM SIGPLAN conference on Object-Oriented Programming, Systems, Languages, and Applications, pages 197–211.

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

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein (2001). *Introduction to Algorithms*. MIT Press and McGraw-Hill, 2nd edition. Chapter 1: Foundations, pp. 3-122.

Enterprise Management Associates (2006). *Get the truth on Linux management*. Online http://www.levanta.com/linuxstudy/, Last accessed at Jan 2007.

Jesus M. Gonzales-Barahona (2006). *The FLOSSMETRICS project* Description of Work. Online http://flossmetrics.org/docs/DoW-2.0-public-0.3.pdf, Last accessed at Jan 2007.

Maurice H. Halstead (1977). *Elements of software science*. Operating and Programming Systems Series, 7.

Sallie M. Henry and Dennis G. Kafura (1981). *Software structure metrics based on information flow*. IEEE Transactions on Software Engineering, 7(5):510–518.

Sallie M. Henry and Dennis G. Kafura (1984). *The evaluation of software systems' structure using quantitative software metrics*. Software: Practice and Experience, 14(6):561–573.

James Howison, Megan Conklin, and Kevin Crowston (2006). *Flossmole: A collaborative repository for FLOSS research data and analysis*. International Journal of Information Technology and Web Engineering, 1(3):17–26.

International Organization for Standardization (2001). *Software Engineering — Product Quality — Part 1: Quality Model*. ISO, Geneva, Switzerland, 2001. ISO/IEC 9126-1:2001(E).

Philip M. Johnson, Hongbing Kou, Michael G. Paulding, Qin Zhang, Aaron Kagawa, and Takuya Yamashita (2005). *Improving software development management through software project telemetry*. IEEE Software, August.

Stephen H. Kan (2002). *Metrics and Models in Software Quality Engineering* (2nd Edition). Addison-Wesley Professional.

Wei Li and Sallie M. Henry (1993). *Object-oriented metrics that predict maintainability*. Journal of Systems and Software, 23(2):111–122.

Zhenmin Li, Shan Lu, Suvda Myagmar, and Yuanyuan Zhou (2006). *CP-miner: Finding copy-paste and related bugs in large-scale software code*. IEEE Transactions on Software Engineering, 32(3):176–192. (doi:10.1109/TSE.2006.28)

Anton Litvinenko and Mark Kofman (2007). *Sourcekibitzer*. Online http://www.sourcekibitzer.org/, Last accessed at Jan 2007.

Martin Lorenz and Jeff Kidd (1994). *Object-Oriented Software Metrics*. Prentice Hall Object-Oriented Series. Prentice Hall.

Thomas J. McCabe and Charles W. Butler (1989). *Design complexity measurement and testing*. Communications of the ACM, 32(12):1415–1425, December.

Thomas J. McCabe and Arthur H. Watson (1994). *Software complexity*. Crosstalk, Journal of Defense Software Engineering, 7(12):5–9, December.

Thomas J. McCabe (1976). *A complexity measure*. IEEE Transactions on Software Engineering, 2(4):308–320, December.

Microsoft (2006). *Get the facts on Windows server system and Linux*. Online http://www.microsoft.com/windowsserver/facts/default.mspx, Last accessed at Jan 2007

Ranjith Purushothaman and Dewayne E. Perry (2005). *Toward understanding the rhetoric of small source code changes*. IEEE Transactions on Software Engineering, 31(6):511–526. (doi:10.1109/TSE.2005.74)

Ralf Reißing (2001). Towards a model for object-oriented design measurement. In *5th International ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering*.

Jason Robbins (2004). *Making Sense of the Bazaar: Perspectives on Open Source and Free Software*, Practices by Adopting OSSE Tools. MIT Press.

Santonu Sarkar, Girish Maskeri Rama, and Avinash C. Kak (2007). *API-based and information-theoretic metrics for measuring the quality of software modularization*. IEEE Transactions on Software Engineering, 33(1):14–32. (doi:10.1109/TSE.2007.4)

Diomidis Spinellis (2005). *ckjm: Chidamber and Kemerer Java Metrics*. Online, Last accessed at January 2007.

Diomidis Spinellis (2006). *Code Quality: The Open Source Perspective*. Addison-Wesley, Boston, MA.

Ramanath Subramanyam and M. S. Krishnan (2003). *Empirical analysis of CK metrics for object-oriented design complexity: Implications for software defects*. IEEE Transactions on Software Engineering, 29(4):297–310. (doi:10.1109/TSE.2003.1191795)

Nikolaos Tsantalis, Alexander Chatzigeorgiou, and George Stephanides (2005). *Predicting the probability of change in object-oriented systems*. IEEE Transactions on Software Engineering, 31(7):601–614. (doi:10.1109/TSE.2005.83)

Davor Cubranic, Gail C. Murphy, Janice Singer, and Kellogg S. Booth (2005). *Hipikat: A project memory for software development*. IEEE Transactions on Software Engineering, 31(6):446–465. (doi:10.1109/TSE.2005.71)

Chadd C. Williams and Jeffrey K. Hollingsworth (2005). *Automaticmining of source code repositories to improve bug finding techniques*. IEEE Transactions on Software Engineering, 31(6):466–480. (doi:10.1109/TSE.2005.63)

M. Xenos, D. Stavrinoudis, K. Zikouli, and D. Christodoulakis (2000). Object-oriented metrics — a survey. In *FESMA'00, Proceedings of Federation of European Software Measurement Associations*, Madrid, Spain.

Thomas Zimmermann, Peter Weissgerber, Stephan Diehl, and Andreas Zeller (2005). *Mining version histories to guide software changes*. IEEE Transactions on Software Engineering, 31(6):429–445. (doi:10.1109/TSE.2005.72)