Requirement Engineering and Study of Software Evolution by Using an Open Source Bug Reporting Tool

Parastoo Mohagheghi[1,2], Reidar Conradi[1,3]

[1]Department of Computer and Information Science
Norwegian University of Science and Technology, NO-7491 Trondheim, Norway,
[2]Agder University College, No- 4898 Grimstad, Norway
[3]Simula Research Laboratory, P.O. Box 134, NO-1325 Lysaker, Norway
parastoo@idi.ntnu.no, conradi@idi.ntnu.no

Abstract

This short paper reports work in progress on planning and studying evolution of a product using Trac as an open source bug reporting tool. We have extended Trac to allow collecting data on requirement and defect types, their origin, and their impact on product characteristics. The solution allows integration of all change data in a single tool and study of different aspects of software evolution. It also supports project management in requirement management, planning of releases and allocating resources.

Problem statement

Most of today's software products are in continuous evolution. Previous releases should be maintained (*corrective maintenance* or repairing faults), while new features are added to the product or its performance is improved (*perfective maintenance*), software undergoes refactoring or redesign (*perfective maintenance* to prevent faults), and is adapted to new environments (*adaptive maintenance*). Sometimes other terms are used for maintenance and evolution activities, or these terms are used differently, making comparison of studies difficult. The term *evolution* is often used to cover maintenance activities while the product is still growing. Earlier studies have tried to study the ratio between different categories of maintenance activities, the origin of changes, or the impact of changes on project schedule or defect-density (see [Mohagheghi et al. 2004]). However, empirical studies on software evolution are few, there are often serious validity concerns due to missing and incomplete data, and the researcher has had no control on data collection (thus bounding the research questions to the scarce available data).

Open Source Software (OSS) products are examples of continuously evolving software products delivered in short releases. Due to free access to data on software code and change logs, evolution of OSS products has been studied in terms of growth of software size, the number of changed or added source code modules, or change logs from CVS and similar version control systems. Among the studies are [Chen et al. 2004] [Capliuppi 2003] [Antoniol et al. 2004] [Scacchi 2004]. While these studies provide valuable information on the rate of growth of products, other questions on the origin of changes (e.g. changed environment, market needs etc.) or type of changes (extensions, perfection, functional vs. non-functional) are difficult to answer. One challenge is to integrate data from several sources such as version control systems, release notes and defect reporting systems. Chen et al. have studied change logs of three OSS products (small and medium-sized products), and compared these to changes in the source code. Their results showed that change log files are incomplete for research purposes due to the relative high percentage of omissions. The authors have also categorized changes as being corrective, enhancement (perfective and adaptive), a code rearrangement (preventive) or a comment change. For two of the products, the majority of changes are enhancements, while for the third one corrective changes dominate. They mentioned that categorization of changes need fairly descriptive change logs and is subjective. Our study of a commercial product [Mohagheghi et al. 2004] also confirmed the need for integrating data on different types of changes to develop models on evolution.

OSS products and many commercial ones use open source bug reporting tools such as Bugzilla [Bugzilla] or Trac [Trac], where a *bug* may be a defect or an enhancement request. In both tools, the severity of a bug may vary from "blocker" to "trivial", or set to "enhancement". Using the severity field for enhancements is not an ideal solution since enhancements may also be valued as critical, highly desirable, or minor cosmetic. However, storing enhancements and defects in the same tool allows bringing different types of maintenance together and collecting data in a consistent way. Such tools allow automatic generation of reports or statistics, are integrated with relational databases and allow search for special keywords or entries.

Extending Trac for management support and study of evolution

In the SEVO project (Software EVOlution in component-based software engineering) [SEVO], we are interested in studying how and why software products evolve. One of the products being followed is a tool for analyzing accessibility of websites. The project is financed by EU and the tool will be an OSS product being developed in three releases by multiple partners across national boundaries. To address requirement engineering of the product and answer research questions on the origin of changes or their impact on product quality, we have decided to extend Trac with new fields. Trac is chosen since adding fields is easy and it is well integrated with Subversion [Subversion], which is an open source version control system selected for the product. The extensions will help project management in communicating requirements to developers, planning milestones, and allocating requirements to developers and system components. Our approach is close to *action research*; done collectively by researchers and developers in repeated cycles of data collection and improving the process of requirement engineering. The character of the project allows research in addition to development (or in fact research is development).

Certain data such as a unique identifier, reporter's name, severity, and component name are included for each report. However, additional data are required to address management needs and research questions. Among the extensions are:

- We have added the possibility to indicate whether an enhancement is mandatory, highly desirable, desirable, optional or cosmetic. This will allow prioritizing requirements based on their value for customers.

- We have added the possibility to indicate the impact of an enhancement or defect on product characteristics such as functionality, reliability, availability, performance etc. This will again help prioritizing requirements by showing the perceived impact on product characteristics. It will also allow deciding test scenarios and categorization of changes for study of evolution.
- Both defects and enhancements will have a history field. For defects, it may indicate whether the defect is in a new part of the product, is a latent defect from a previous release, is a result of redesign, or is injected after a fix. Such data helps to evaluate the quality of a release and the efficiency of verification techniques in detecting defects. For enhancements, the history will indicate whether it is a new requirement, a modified requirement, a request for redesign or adapting to a new environment. It will allow classification of evolution types and generating metrics such as requirement instability or requirement creep.

Existing fields in Trac such as “affected components”, “milestone to solve” and “assigned to” are used to evaluate the impact of defects or enhancements, plan future releases, and assign these to teams.

Benefits of the solution and conclusions

The benefits of the approach (in addition to following the status of defects and normal corrective tasks) are:

- Using a single available tool to get an *overview of all changes* made to a release in terms of number and type of changes.
- Managing requirements and defects, assigning resources and planning milestones; thus *management support* for evolution planning.
- Providing *traceability* in accordance to the “IEEE recommended practice for software requirements specifications” (IEEE std 830-1998). Each requirement will have a unique identifier, and the origin and impact of a requirement are specified. It is also desirable to ascertain the complete set of requirements that may be affected by modifications in design (or redesign).
- *Evaluating data* needed to study evolution types and requirement evolution over releases. This knowledge will be used to refine research questions and plan data collection in other projects.

We should also study how data from Subversion (such as change logs and software modules checked in for a change) can be integrated with data from Trac to study the impact of changes on source code. Probably, the project should agree on rules such as including Trac identifier in change logs and checking in modules with one change at a time. The extended Trac system has been in use for a few weeks in the first release and we will follow its use over time.

References

[Antoniol et al. 2004] Antoniol, G., Di Penta, M., Gall, H. and Pinzger, M.: Towards the Integration of Versioning Systems, Bug Reports and Source Code Meta-Models. Preliminary version on http://seal.ifi.unizh.ch/fileadmin/User_Filemount/Publications/giulio-setra04.pdf

[IEEE] IEEE Std 610.12-1990 (R2002), http://standards.ieee.org

[Bugzilla] http://www.bugzilla.org/

[Capliuppi 2003] Capiluppi, A.: Models for the Evolution of OS projects. *Proc. Int'l Conference on Software Maintenance (ICSM'03)*, pp. 65-74.

[Chen et al. 2004] Chen, K., Schach, S.R., Yu, L., Offutt, J. and Heller, G.Z.: Open Source Change Logs. *Empirical Software Engineering*, 9(2004), pp. 197-210.

[Mohagheghi et al. 2004] Mohagheghi, P. and Conradi, R: An Empirical Study of Software Change: Origin, Acceptance Rate, and Functionality vs. Quality Attributes. *Proc. of the ACM-IEEE International Symposium on Empirical Software Engineering (ISESE 2004)*, IEEE CS Order Number P2165, pp. 7-16.

[Trac] http://projects.edgewall.com/trac/

[Scacchi 2004] Homepage of W. Scacchi: http://www.ics.uci.edu/~wscacchi/

[SEVO] http://www.idi.ntnu.no/grupper/su/sevo/index.html

[Subversion] http://subversion.tigris.org/