Improving Maintenance through Development Experiences*

Xiaodong Liu, Hongji Yang and Hussein Zedan
Department of Computer Science,
De Montfort University,
England
Email: xdl, hjy, zedan@dmu.ac.uk

Abstract

An empirical comparison is made between present approaches/tools of software maintenance and development. The conclusion is that development technology is more mature in many aspects, such as automation, formalism, confidence, understanding of original systems and application domain. Aims to improve the weakness of maintenance, a unified re-engineering approach is proposed which is based on a wide spectrum language and a set of formal abstraction and transformation rules. A relevant tool is under construction.

Keywords : software maintenance and development, re-engineering, wide spectrum language, formal methods, object oriented, time critical systems.

*The paper is accepted by the Workshop on Empirical Studies in Software Maintenance (WESS98), Metropolitan, Washington D.C., USA, November 1998.

1 Introduction

Software maintenance is attracting more and more attention as it has become evident that old architectures severely constrain new design, which leads to demands for changes to existing software, for instance, fixing errors, adding enhancements and making optimisations. The implementation of the changes themselves creates problems over and above those that are being rectified.

However, the approaches/tools of maintenance are rather weak when contrasted to those of development. Two reasons account for this situation:

- The research and practice of development are more mature than those of maintenance.
- Maintenance is more difficult and with large cost.

Early systems tended to be unstructured and *ad hoc*, which makes it hard to understand their behaviour. System documentation is often incomplete, or out of date. With current methods, it is often difficult to retest or verify a system after a change has been made. Successful software will inevitably evolve, but the process of evolution will lead to degraded structure and yet greater complexity.

Under most situation, the most reliable *corpus* of information on software being maintained for any significant period is the code itself. However, to software engineers, code is more difficult to understand than specification. This accounts for the present backward situation of maintenance to some extent.

Based on the experience of using various maintenance and development tools, in particular Maintainer's Assistant (MA)[1, 6] and SPIN [3], we made a comparison between the current technology of maintenance and development. The result is given in section 2. Our conclusion is that development research has a longer history and is more advanced in a wide span. With the suggestion of successes in development technology, a unified approach for system re-engineering is proposed in section 3. Section 4 summarises the experiment test result with case studies.

2 Empirical Study of the Technology of Maintenance and Development

We based our study mainly on two tools: MA and SPIN. MA is an interactive tool which helps the user to extract a specification from an existing source code program. It is based on a program transformation system, in which a program is converted to a semantically equivalent form using proven transformations selected from a catalogue. Spin is a widely distributed software package that supports the formal verification of distributed systems. The software was developed at Bell Labs in the formal methods and verification group. It uses Linear Temporal Logic as its base.

The aim of the ReForm project is to create a code analysis tool–the **Maintainer's Assistant**, aimed at helping the maintenance programmer to understand and modify a given program. Program transformation techniques are employed by the Maintainer's Assistant both to derive a specification from a section of code, and to transform a section of code into a logically equivalent form. The aim is to provide a tool with features such that:

- It acts, initially, on existing program code as a tool to aid comprehension (possibly by producing specifications);
- Only the program code is required;
- The system can work with any language by first translating—with a stand-alone translator–into WSL;

- Changes are made to the WSL program by means of transformation;
- The system incorporates a large, flexible catalogue of transformations;
- The applicability of each transformation is tested before it can be applied;
- The system is interactive and incorporates an X-Windows front end and pretty-printer called the Browser;
- The system includes a database structure to store information about the program being transformed, such as the variables assigned to within a given piece of code;
- The system includes a facility to calculate metrics for the code being transformed.

One of the most important successes of Maintainer's Assistant is that it is based on a wide spectrum language whose syntax and semantics are formally defined. Maintainer's Assistant is a successful case of applying wide spectrum languages in re-engineering area. However, Maintainer's Assistant focused on transformations rather than abstraction. It involved very little in how to use multi-leveled abstractions and relevant abstraction rules to reach a good system re-engineering, especially reverse engineering. The Wide Spectrum Language in Maintainer's Assistant is sequential and non-timed, which limits its application domains such as real-time systems.

As a formal methods tool, SPIN aims to provide:

1. an intuitive, program-like notation for specifying design choices unambiguously, without implementation detail.
2. a powerful, concise notation for expressing general correctness requirements, and
3. a methodology for establishing the logical consistency of the design choices from 1) and the matching correctness requirements from 2).

The design methodology that is supported by SPIN can be summarised as follows:

1. A distinction is made between behaviour and requirements on behaviour. The designer specifies the two aspects of the design in an unambiguous way by defining a verification or prototype in the language PROMELA.
2. The prototype is verified using the model checker SPIN. The requirements and behaviours are checked for both their internal and their mutual consistency.
3. The design is revised until its critical correctness properties can successfully be proven. Only then does it make sense to refine the design decisions further toward a full systems implementation.

The increasing number of applications of SPIN, and the growing acceptance of formal methods in general, are hopeful signs that this paradigm of design is maturing, and is gaining recognition where it counts : among the practitioners of distributed systems design.

The conclusions is that development technology is more advanced than that of maintenance in many aspects. We present our study according to the following criteria:

1. Maturity. Development is far more well studied and has more approaches and tools than maintenance. The theory foundation of Spin has been well developed for a rather long history and has a wide and strong background. MA, although a successful example in maintenance, is weaker in both theory and practice when compared with Spin.

2. Application Domain. Most maintenance activities are still focused on sequential procedural systems, while many development activities already dip into special domains, such as time-critical, parallel/concurrent systems. Spin aims for concurrent distributed real-time system, while MA aims mainly at sequential non-time applications.

3. Formalism. Most maintenance approaches and tools use ad hoc technology instead of formal methods. This limits their confidence and automation. On the other hand, formalism is already popular in development approaches and tools. Spin is based on Linear Temporal Logic and theorem proving theory. MA although has a well defined formal semantics foundation, it is still weaker than Spin. And most other maintenance tools only use ad hoc technology.

4. Objectives. Maintenance aims to recover design and specification from old code, to improve it and then to move down to a new system; while development always moves from specification to source code. As stated before, Maintainer's Assistant aimed at helping the maintenance programmer to understand and modify a given program. Spin supports the formal specification and verification of distributed systems, time-critical systems.

5. Key activity. In maintenance the key activity is abstraction; while in development it is refinement.

6. Understanding of System. This criterion is important both to maintenance and development. However, the target system is normally better understood by the software engineer during development than maintenance since specification is easier to understand than source code. When using MA, you actually has to read and understand the processed code first, whilst the specification in Spin is more high-leveled and easier to understand.

7. Automation. Maintenance tools are usually weak in automation in contrast with development tools. Many factors affect automation, for instance, using of formalism, rules to fulfill refinement or abstraction, understanding of the processed system (code or specification). MA needs much more human intervene than Spin when using them for applications, especially using it to do abstraction.

8. Industrial Scale. Only a few maintenance tools bear the ability to deal systems in industrial scale; while relatively 'many' development tools have this ability. Spin can manage more large-scale application than MA.

9. Confidence. Maintenance approaches/tools tend to be less accurate to original systems when contrasted to development approach/tools. Lack of formalism and difficulties in understanding original systems contribute to this result. When using MA for program transformation, the confidence of the result is better than most of other maintenance tools, although sometimes it still produces "unreasonable" result.

10. Emergency in need: Both efficient maintenance and development approaches/tools are needed in practice.

3 Proposed Approach and Tool

From the empirical study in section 2, we conclude that maintenance is weak in automation, formalism, confidence, understanding of original systems and application domain. Maintenance needs further research. In order to overcome these incompetences, it is helpful and promising to develop a formal approach accommodating reverse engineering systems with an emphasis on abstraction and also a provision of incorporating Object-Oriented technique for re-engineering these systems. With such an approach, the confidence and automation of maintenance activity will be increased. A formal specification will be extracted from source code through abstraction rules to help the software engineer get a good understanding of original systems, which facilitate maintenance in a great extent.

The approach is based on the construction of a wide spectrum language, known as R-WSL, and a set of abstraction and transformation rules, which enjoys a sound formal semantics. A spectrum of abstraction level from code to specification and abstraction rules to cross these levels are developed and defined in Interval Temporal Logic [4, 5, 2]. Besides general sequential systems, our approach treats concurrent real-time systems as its specific domain.

A prototype tool named Re-engineering Assistant is being built based on the proposed framework, as a re-engineering workbench. We have also developed metric measures for measuring abstractness in such a tool.

4 Case Studies

The approach has been applied on small to medium case studies, and the result shows it is very promising and worth further development.

Typical finished case studies include: (1) Robot Control System: non-timed, sequential system; (2) Mine Drainage System: time-critical, interrupt-driven system. (3) Task Farm System: parallel system with synchronous and asynchronous communication.

5 Conclusion

Development and maintenance are two different but closely related activities. It is helpful to compare, contrast and borrow successful ideas between them. Since development has a longer history than maintenance, it often appears as technologies originated in development find their new place in maintenance.

What is required is a reliable link between development and maintenance methodologies. This will ultimately result in a highly and tightly integrated workbench which both deal with development and maintenance.

References

1. Bennett, K. H., Bull, T., and Yang, H. A transformation system for maintenance — turning theory into practice. In *IEEE Conference on Software Maintenance-1992* (Orlando, Florida, Nov. 1992).

2. Cau, A., Zedan, H., Coleman, N., and Moszkowski, B. Using ITL and tempura for large scale specification and simulation. In *Proceedings of 4th EUROMICRO Workshop on Parallel and Distributed Processing, IEEE* (Braga, Portugal, 1996), pp. 493–500.

3. Holzmann, C. J. *Design and Validation of Computer Protocol.* Prentice-Hall International, 1990.

4. Moszkowski, B. *A Temporal Logic for Multilevel Reasoning about Hardware.* IEEE Computer Society, Feb. 1985.

5. Moszkowski, B. *Executing Temporal Logic Programs.* Cambridge University Press, Cambridge UK, 1986.

6. Yang, H., and Bennett, K. H. Extension of A transformation system for maintenance — dealing with data-intensive programs. In *IEEE International Conference on Software Maintenance (ICSM '94)* (Victoria, Canada, Sept. 1994).