**Title**  
System Documentation for Software Maintenance

**Research Area or Sub-area**  
Reverse Engineering (RE) software systems (Research area: Software Maintenance).

**Problems and Importance of the Study**  
Until today, maintenance is seen as the most costly phase in a software life cycle. Problems in software maintenance relates to how much information is available from the documents of the software especially with respect to the architecture and design of a system. The study by Vliet (2000) stated that documentation being absent, out-of-date or at best insufficient as the 3rd major cause of maintenance problems. While the survey by Sousa and Moreira (1998) found that the lack of documentation of applications was in the 2nd ranking of the three biggest problems related to software maintenance process. Generally, low quality documentation will contribute to software maintenance problems.

There are three major problems highlighted in my study. The problems are: 1) Low quality, incomplete and not standardized documentation, 2) Unreliable and out-of-date documentation, 3) Computer-Aided Software Engineering (CASE) tools not producing documentation as expected.

1) **Low quality, incomplete and not standardized documentation**  
Organization that does not implement CASE tools in their software development needs to produce the system documentation manually. Programmers need to read through all the source codes again and transform them into graphical notation. “This can be a tedious job, which requires considerable concentration. It is all too easy for concentration to collapse, for even a very short period and miss some vital piece of information” (Lincoln, 1993). Besides, a lot of programmers only start the documenting activity after software development has been completed. Time constraint and commercial pressures or deadlines could be the factors contributing to the problem (Vliet, 2000). Programmers also dislike documenting system as this is seen as rather boring task compared to the excitements of creation in design and implementation (Macro, 1990). In addition, some organizations do not apply any documentation standards or provide guidelines to programmers who produce system documentation. Hence documentation produced in such organizations is of different format. All the mentioned factors might lead to low quality documentation.

2) **Unreliable and out-of-date documentation**  
Software evolves and thus always needs to be maintained due to new requirements, changes in environment or other factors. Software maintainability is extremely vital to ensure long lasting software evolution. Pigoski (1997) indicates that documentation is one of the important factors in software maintainability. Nevertheless, only the documentation for the first maintenance could be reliable. The link between a program and its associated documentation are sometimes lost during the maintenance process and this may be the consequence of poor configuration management or due to adopting a “quick fix” approach to maintenance (Sommerville, 1997). Hence, the documents become out-dated and unreliable for reference in the following maintenance process.

3) **CASE tools not producing documentation as expected**  
There are a lot of CASE tools of different classes that can assist software engineers throughout the whole Software Development Life Cycle (SDLC). As for maintenance all

---

1
tools might be used again while repeating the SDLC. Such fully integrated CASE tools will normally produce a self-generated documentation that will be very useful to maintainers. However, most organizations do not use CASE tools to support all phases of SDLC. “These reasons range from a lack of vision for applying CASE to all aspects of the SDLC to the belief that CASE technology will fail to meet an organization’s unique system development needs” (Hoffer, et al., 1999). Hence, system developers might just use a particular CASE tool to draw the diagrams that capture user requirements in the analysis and design phase of SDLC but not using code generators or RE utility provided. Therefore, there is a lack of integration among the system documents and the source codes. Consequently, the document produced by a document generator is not sufficient anymore. Although a number of RE tools have been developed to produce parts of documents automatically, they are not widely used because (Canfora, et al., 1991): 1) Documents generated are either too general or too detailed, 2) Documents produced lack the data the maintainer needs and 3) Inflexibility of RE tools.

Hence, this research is intended to deal with the questions that focus on the system documentation, which is vital to software maintainers. The main question is “How to produce a system documentation that is reliable to software engineers who are going to be involved in the future software maintenance?” The sub questions are: 1) Why system documentation produced/maintained is always of low quality, incomplete, not standardized, unreliable and out-of-date, 2) Why some CASE environments which incorporate documentation generator utility are not able to produce system documentation as expected by maintainers, 3) Why RE tools are not able to overcome the problems in producing system documentation, 4) What are the important features in system documentation which must be captured since they are critical for the subsequent maintenance process, 5) How to manage the software configuration management associated with continuously evolving documents to facilitate maintenance and 6) How to verify that a particular methodology in document generation and maintenance is effective and useful.

Software maintenance incurs around 90% of software life-cycle costs in early 1990s (Moad, 1990), which is extremely high. Hence this problem merits serious investigation. It is expected that this study will be beneficial to other researchers in software maintenance field and software maintainers who will use the prototype tool to be developed in this research.

Other Related Works
Since the early 1990s, CASE (Computer-Aided Software Engineering) vendors had discovered that by reengineering or redeveloping with CASE tools the “software maintenance crisis” would be resolved (Pigoski, 1997). One important technique required in a reengineering process is reverse engineering (RE) which refers to the process of creating design specifications for a system or program module from source codes and data definitions (Hoffer, et al., 1999). One technique closely related to RE is program comprehension. Thus a lot of researches in program comprehension have been and are being conducted in order to improve program comprehension. Since Weiser (1984) introduced the approach of program slicing in program comprehension, a number of researches have been based on his work. For instance Rilling (1998) investigates dynamic slicing and its application in program comprehension while Song (1999) introduces forward dynamic slicing in his research. Recent work by Villavicencio (2001) promotes a technique based on the automatic comparison of slices that allows analyst to focus his attention on a meaningful code for the design of program plans. Other approach in the work of Abd-El-Hafiz and Basili (1995) introduces a knowledge-based approach to the automation of program understanding that
focuses on loops in programs. Canfora, et. al. (1999) proved that by using eclectic approach
where a domain expert software engineer tailor and combine existing approaches may
overcome the limitation of the single approaches. Cox and Clarke (2000) suggest an
alternative using hierarchical lexical analysis augmented with simple data structures if
traditional parsing techniques cannot parse the code to obtain syntactic level models of code
being maintained. In addition, a lot of tools are available that apply certain approaches in
program understanding during reverse engineering process. Some instances of research
prototypes are Rigi, Maintainer’s Assistant and PBS while commercial tools are SNiFF+,
Logiscope and Imagix 4D. A number of researches evaluated the tools such as Bellay and
Most of the evaluation works show that no tool is “perfect” in producing system
documentation needed in a software maintenance process, which normally involve different
languages, platforms and types of information.

Hence my study will attempt to provide a document generator that can produce a system
documentation (for the sake of software maintenance) starting from development stage,
which is considerably complete with required level of abstraction either in graphical or
textual form. The tool should be able to keep track the changes made to the system in order
to ensure up-dated link between source codes and system documentation. The tool is
expected to work with procedural or structured programming languages of non-VDE (Visual
Development Environment) such as COBOL, Clipper and C. Besides, the tool will be
designed to be flexible enough to suit with more languages and platforms.

**Research Design and Proposed Solution**
The research design is briefly indicated below:

- **SU**: Software Understanding, SU = {High, Medium, Low}
- **SWD**: Software developer
- **SWM**: Software maintainer
- **SD**: System documentation (Independent variable)
- Pre-requisite: SD must exist
- i : Development stage, i = 1 (software is developed once the rest will be under
software maintenance) while j : Number of maintenance, j ∈ {1, 2, 3, ..., n}.

**Hypotheses H₁ij**: The SU has significant improvement by applying RE tool to produce SD
starting from Development Stage i until jth number of maintenance process (not only when
software has become legacy system or only during software maintenance process).

**Hypotheses H₀ij**: The SU has no significant improvement although by applying RE tool to
produce SD starting from Development Stage i until jth number of maintenance process (not
only when software has become legacy system or only during software maintenance process).

It is believed that the problems in producing system documentation mostly occur right from
the software development stage. Thus our study would like to propose the use of RE tool to
produce system documentation starting from the development stage to the following software
maintenance processes not only when the system becomes a legacy system.

**Expected Contribution and Work Progress**
The prototype RE tool that is going to be developed and tested according to the research
design is expected to be able to lighten the burden of software maintainers who partially use
or do not use CASE tools in SDLC. A miniature pilot study has been conducted in order to
verify the questionnaires of a survey on software engineers who develop or maintain software systems. The pilot study will provide a guideline to refine the questionnaire for the actual survey. The goals of the survey are to identify the problems in producing or maintaining system documentation, CASE tools usage particularly RE tools and expectation of software engineers towards RE tools that generate system documentation. A comparison of tools (Sulaiman, et. al. 2002) based on the type of information and its abstraction required in relation to maintenance categories was already conducted. The study shows that there is a relationship between the type of information and its abstraction with the category of maintenance task.

Research Methodology

The miniature pilot study is being carried out through the Internet. The respondents are software engineers of various demographic. The actual survey will be carried out in a number of selected organizations in Malaysia (and around Asia). An industrial observation was already done in order to identify the types of information required. More observations will be carried out in the near future. Upon completing the prototype tool, a pilot test will be conducted on a group of computer science or software engineering students who have at least moderate background in programming. The design will be based on the research design mentioned before. Case studies of applying the prototype tool in industrial background will be mandatory in order to verify the effectiveness and weaknesses of the tool.

References


Song, Yeong-Tae (1999). “Forward Dynamic Slicing Software Systems”. Faculty of the University of Texas at Dallas: PhD Dissertation Canada: UMI.


