Integrated Functional and Executional Modeling of Software Using Web-Based Databases

Deepak Kulkarni  
NASA Ames Research Center, USA

Roberta Blake Marietta  
Caelum Research/NASA Ames, U. S. A.

To preserve safety during software maintenance, an engineer needs to be able to answer a variety of questions about the software in a very short time. In the current state of practice, this is sometimes not possible. Any solution to this problem should allow group usability of information, automatic creation of information in target form from the structure in which it was created and selective fast access. This paper presents a framework for doing so based on integrated modeling of software, use of automatic information extraction tools, web technology and databases. We then describe demonstrations that indicate that such a solution can be produced at a reasonable cost in an error-free manner and would meet the concerns of safety analysts about the sensitivity of safety analysis information.

NASA’s software subsystems undergo extensive modification and updates over their operational lifetimes. It is imperative that modified software should satisfy safety goals. To ensure this, a maintainer needs to quickly understand safety issues concerning a modification. This report discusses the difficulties encountered in doing so and proposes a solution based on integrated functional and executional modeling using web and database technologies. In the next section, we discuss this problem in more detail. In the third section, we propose a solution to the problem. In the fourth section, we describe demonstrations of the concepts. The final section is our conclusion.

The Problem of Understanding Safety Issues During Maintenance

The steps in the maintenance process include understanding the existing code, determining where to make changes, assessing the impact of changes, rebuilding code after changes, and regression testing to validate changes. Safety issues have to be addressed in each of these steps. As the last two impact safety indirectly through reliability, we will only focus on the first three in this report.

Understanding existing code, which consumes the majority of time during maintenance (SEI, 1995), involves understanding the safety aspects of existing code, among other things. Second, one needs to determine what to change and what to reuse when a hazard, safety requirement or safety assumption is added or altered. The determination of what to change may also involve choosing between alternatives by evaluating which change is safer to make. Finally, one must assess the impact of changes on the safety aspects of the software: functionality/requirements/hazards. Failure to do so has been a significant cause of software accidents (Leveson, 1995). Below is a list of questions that may need to be answered to preserve safety during the maintenance phase:

Questions related to execution flow:

• Which code-level entities will become safety-critical / non-critical as a result of the change?
• What is the impact of changes on the testability of faults in safety-critical software?
• Which modules would be impacted as a result of a change in a module?
• Which assumptions (tests) need to be considered in making a change?
• What module may have caused a test to fail?

Questions related to functionality:
• What modules/assertions implement a safety functional requirement being changed?
• What safety requirement and Preliminary Hazard Analysis is satisfied by a module and what is the associated hazards analysis?

Questions related to both functionality and execution flow:
• What is the safety requirement functionality associated with all causative/impacted modules?
For example, module C is modified making it slower to execute. The execution flow is A BBB C and the safety functionality for A, B, C and the whole software includes timing constraints. Thus, both information derived from execution flow and functional modeling can be used to conclude if the change is acceptable or could cause a hazard.
• What safety-critical components may be affected by a change?
• What changed components may have caused a fault?
• What safety-critical components may have caused a fault?

The Difficulty of Safe Maintenance

The difficulty in getting the answers to such questions during the maintenance phase is that a huge amount of information is generated during development of the software, and the maintainer can only justify spending very limited time in answering any questions. To illustrate this, we would like to give an example of a joint NASA-Rockwell project with about 30K lines of code and a dozen data files that were to have configuration information. The software made almost 80 or so assumptions about the nature of the information in the data files. A person modifying the data files would typically make a small change in the file, sometimes allocating just a few minutes of his time for the activity. Clearly, there was not enough time to read through a large document with 80 assumptions and other information, were such a document to exist. Furthermore, the document itself was not created as the requirements kept on evolving and there was no quick way to update such a document in the face of continuous changes. While updating the document would have been time-consuming, fixing numerous errors resulting from changes consumed far more time.

A further problem is that information is created and analyzed during different phases using different tools. For example, a Fault Tree Analysis tool may be used to create a safety Fault Tree Analysis, whereas source code may be created using Computer-Aided Software Engineering (CASE) tools. Maintenance queries may need to access information created by both the fault tree and the source code to answer some of the questions outlined above. Other tools that may be used in software development are editors, compilers, cross-references, debuggers, performance analysis tools, version management tools and testing tools. Some of these may produce data that would also be of relevance to the questions above.

Another problem is that the data needed to answer some maintenance questions is owned and controlled by diverse groups and departments. An example process at NASA Ames (Figure 1) may consist of a System Requirements document being developed by a Systems Analyst. A safety analyst then develops Hazard Reports based, in part, on these system requirements. A Software Designer develops a software architecture or design based also upon the requirements specification. An individual programmer develops source code based upon the software design. In this case, a system requirement is a common thread which can be used to relate the various data sources and develop an integrated data model. With an integrated model, specific individuals or groups continue to be responsible for controlling their own data, but have no control over their data’s relationship to other data. The unrestricted availability of data may, in some cases, create problems that are worse than the benefits derived from the technology. For example, if a hypothetical accident scenario becomes available to newspapers, it may result in the cancellation of a project.

General Constraints on the Solution Approach

In general, a solution to the safe maintenance problem needs to meet the following requirements:

1. Automatic Creation the of Model: Given the frequency of change of software entities (daily or weekly) and the volume of information involved, manual translation of