THREE CHALLENGES IN PERFORMING RELIABLE SOFTWARE MAINTENANCE ON AUTOMOTIVE CONTROL SYSTEMS

Shihong Huang
Computer Science & Engineering
Florida Atlantic University
shihong@cse.fau.edu

Scott Tilley
Department of Computer Sciences
Florida Institute of Technology
stilley@cs.fit.edu

Key Words: software maintenance, automotive control systems, reliability

Abstract

Modern automotive control systems are incredibly intricate hybrids of software components and hardware mechanisms. Gaining a sufficient understanding of such a system in order to perform reliable maintenance with predictable results can be quite difficult. This paper discusses three specific challenges related to this task: (1) having tools that provide sufficiently robust analysis to support the decision-making process of engineers performing the maintenance; (2) managing the complexity of mixed hardware/software analysis (by both the user and the tool); and (3) constructing tools that are suitable for adoption by the practitioner community so that they can provide an effective automated aid. These three challenges were identified as part of our own experience in working on projects in this application area.

1. Introduction

Maintaining legacy software systems has long been known to be an exigent task. Lack of proper documentation, the inherent complexities of large-scale and long-lived applications, and the dearth of effective tool support have plagued software engineers for many years. As the amount of software embedded in many of today's commercial products grows, the maintenance problem grows seemingly unabated.

One application area that poses singular challenges and that is becoming increasingly important is embedded control systems in automobiles. Modern vehicles are now dependent on software-intensive subsystems to control many aspects of the vehicle's operation. For manufacturers of premium brands, software is often relied upon to provide customer-driven features that differentiate the final product from the mass-produced competition. The result is hybrid control systems that are exceptionally difficult to engineer and similarly hard to reliably maintain.

In our experience working with several industrial partners in performing software maintenance of real-world control systems, we have found that reliability of maintenance is a direct result of the design quality of the tools used in performing the analysis tasks. The analysis can be performed using reverse engineering [2], which is a process of gathering data through automated processing of the system (e.g., parsing source code) and generating abstract representations of the system to support program understanding.

The reverse engineering tools employed in support of maintenance of automotive control systems have unique requirements imposed on them due to the intricacies of the application domain. The next three sections of the paper discuss three such requirement challenges: robustness, mixed hardware/software analysis, and adoption. Each of these challenges represent research opportunities for academic software engineers hoping to have their results adopted by practitioners. The last section of the paper also briefly discusses further work in the area, with particular attention to interdisciplinary education.

2. Challenge 1: Robustness

Performing analysis of embedded automotive control systems induces a requirement of robustness on the reverse engineering tool or technique being used. For example, full source code analysis is often needed to support informed decision-making regarding changes made to the software. Many reverse engineering tools only support a subset of the programming language, leaving out complex constructs such as pointer alias analysis. Unfortunately, supporting the complete language is needed for real-world applications. Otherwise, the resultant information is incomplete, leading to possibly erroneous conclusions regarding issues such as control pathways and variable usage.

Another example of robustness is the need to perform low-level macro traceability. Support for macros is usually implemented in the pre-processor stage in a traditional compiler. However, separating macro processing from language processing for program understanding purposes can have negative consequences. The main drawback is the lack of traceability between the definition of a macro, such as a bit variable access mechanism that are common in control systems, and its usage throughout the code. Without such traceability, the software engineer only sees the expanded source code, not the original definition, which burdens the user with the need to remember the correspondence between the macro's logical name and its expanded source code construct. The result may be an unreliable change to the system, with potentially disastrous consequences.

A third requirement is the ability to identify platform-specific code patterns. Such patterns can be used to guide future refactoring of the application for purposes such as porting it to a new hardware platform and/or operating system. Identification of such patterns requires that the reverse engineering tool support the previous two requirements of full source code analysis and macro traceability, but it must also have a library of code clichés available to match against specific instances located in the control system. This is a difficult task for even the best reverse engineering tool, but satisfying it can help the next challenge: managing mixed hardware/software analysis.

3. Challenge 2: Mixed HW/SW Analysis

A defining characteristic of automotive control systems is that they are complex mixtures of hardware and software. Control systems are often distributed, real-time applications that link together multiple processing units across diverse network topologies using various communication protocols. They are also composed of products originating from different vendors, which means complete control over system quality is not always possible.

To function effectively, control systems must have their respective software and hardware parts functioning harmoniously. Sometimes the differentiation between hardware and software is not entirely clear, since firmware and emulators blur the distinction between the two. This implies that a reverse engineering tool must accommodate mixed hardware/software analysis of the system's constituent components.

For example, the execution of source code components can be triggered by a hardware-driven event, such as an interrupt or an external sensor. From a purely software point of view, such a component is the root of a potentially complex call sequence, but one that may not have an identifiable entry point. Only by connecting the hardware trigger to the software component can the total system be modeled and hence analyzed.

Mixed hardware/software analysis also imposes challenges on the person performing the maintenance. It is already quite difficult to fully master modern software applications [4]. When the nuances of hardware devices are added to the system, the tasks becomes even more substantial. Since not every software engineering has deep knowledge of the hardware aspects of a control system, and not every electrical/computer engineer (say) has broad knowledge of modern software engineering practices, bridging these two disciplines is demanding; proper tool support can help in this regard. However, such support is only realized if the tools are actually used and adopted in daily usage.

4. Challenge 3: Adoption

The third challenge in performing reliable software maintenance on automotive control systems is more properly stated as a (mostly unmet) need of the person who is working on the application: having proper tool support. "Proper" in this context connotes adjectives such as "usable", "effective", and "robust". The tools should also be easily integrated into existing processes.

Many reverse engineering tools that are available have their beginnings as research prototypes [5]. Such tools invariably suffer from a lack of maturity when compared with commercial offerings. This leads to difficulty in convincing practicing software engineers to adopt the solution: it must work as expected, in real-world settings, in a consistent manner. Sadly, many research-oriented tools require considerable tweaking, handholding, and on-site support, making the tool all but useless except for the most technologically adept user.

Academics and researchers need to be cognizant of the industrial requirements for their tool or technique. Such requirements can be as simple as functioning on a specific platform, or interoperating with other tools already deployed at the customer's site. These seemingly minor details can in fact make the difference between a promising tool that fails to be adopted, and a widely used solution that enjoys considerable community support.

There is at least one more step that tool developers should take to foster the adoption of their efforts: they should provide objective evidence of the tool's efficacy, for example by performing empirical studies [3]. For too long the software engineering community has been forced to accept statements of a tool's usefulness based solely on faith, on the energetic words of the tool's developers. Thankfully, there is now a concerted effort to remedy the situation, to instill in the tool developer a sense of the need for an evidence-based approach to technology adoption [1].

5. Summary

This paper presented three challenges in performing reliable software maintenance on automotive control systems. The first challenge was robustness of the reverse engineering tool used in performing system analysis. The second challenge was mixing hardware/software analysis in a holistic manner. The third challenge was constructing the reverse engineering tool (or the underlying technique) so that practitioners can easily adopt it. Without adoption, tools will not be used, and the resulting maintenance will be ad-hoc and unreliable.

One area of future work that we see as of paramount importance is cross-disciplinary education. As stated earlier, automotive control systems are an interesting mix of hardware and software. Becoming adept with such systems requires mastery of concepts from several fields. However, the problems present in the area are giving rise to interesting research opportunities.

References

[1] Budgen, D.; Hoffnagle, G.; Müller, M.; Robert, F.; Sellami, A.; and Tilley, S. "Empirical Software Engineering: A Roadmap." *Proceedings of the 10th International Conference on Software Technology and Engineering Practice* (STEP 2002: Oct. 6-8, 2002; Montréal, Canada), pp. 180-184. Los Alamitos, CA: IEEE Computer Society Press, 2003.

[2] Chikofsy, E.; and Cross, J. "Reverse Engineering and Design Recovery: A Taxonomy." *IEEE Software* 7(1):13-17, January 1990.

[3] Huang, S. and Tilley, S. "On the Challenges in Fostering Adoption via Empirical Studies." *Proceedings of the 4th International Workshop on Adoption-Centric Software Engineering* (ACSE 2004: May 25, 2004; Edinburgh, Scotland, UK). May 2004.

[4] Tilley, S.; Gerdes, J.; Hamilton, T.; Huang, S.; Müller, H.; Smith, D.; and Wong, K. "On the Business Value and Technical Challenges of Adopting Web Services." *Journal of Software Maintenance and Evolution: Research and Practice*, 16:31-50. John Wiley & Sons, 2004.

[5] Tilley, S.; Huang, S.; and Payne, T. "On the Challenges of Adopting ROTS Software." *Proceedings of the 3rd International Workshop on Adoption-Centric Software Engineering* (ACSE 2003: May 9, 2003; Portland, OR), pp. 3-6. Published as CMU/SEI-2003-SR-004. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, June 2003.