Issues in the Assurance of Component-Based Software

Gary Vecellio
The MITRE Corporation
1820 Dolley Madison Blvd.
McLean, VA 22102 USA

+1 703 883-5857
vecellio@mitre.org

William M. Thomas
The MITRE Corporation
1820 Dolley Madison Blvd
McLean, VA 22102 USA

+1 703 883-6159
bthomas@mitre.org

1 INTRODUCTION

Software is increasingly being used in systems where the consequence of failure is high. For example, software is being used in systems that threaten life, health, national security, the environment, and the economy. Also, in an effort to decrease software cost and speed time-to-market software developers are increasing their use of COTS software. Operating systems, middleware, software components, and software frameworks are being purchased off-the-shelf and included in critical applications. Yet there are few if any ways to determine adequate software components from inadequate ones. Nor are there adequate ways to determine the effect that replacing software components will have on the system's behavior.

One of the recommendations of the President's Information Technology Advisory Committee (PITAC) report is to "fund more fundamental research in software development methods and component technologies." The report identifies the need for "Software methods for efficiently creating and maintaining high-quality software of all kinds and for ensuring the reliability of the complex software systems that now provide the infrastructure for much of our economy" [5]. Specifically the report calls for research to explore and create:

- component-based software design and production techniques, and the scientific and technological foundations needed for a software component industry,
- techniques for using measurably reliable components and their aggregation into predictably reliable and fault-tolerant systems,
- theories, languages and tools that support automated analysis, simulation, and testing of components and their aggregation into systems, and
- techniques for aggregating provably secure components into provably secure systems.

The concern over the use of COTS software components in critical systems is echoed elsewhere. For example, The National Research Council's report [10] states:

> *"Commercial software packages and systems - and not systems custom-built from scratch - are a central subject of this report... Research that ignores COTS software could have little impact on trustworthiness for future Networked Information Systems. In the past, computer science research programs serving military needs could safely ignore commercial software products; that course now invites irrelevance."*

Recent advances in component technology are making COTS solutions even more attractive. The increasing use of COTS in critical and complex systems is inevitable. Continued research is needed to assure that such systems meet their goals. We are currently investigating issues associated with assurance of This paper reviews standards for COTS software in critical applications, identifies new challenges associated with component based COTS software, and suggests some static analysis approaches to help address the these challenges.

2 STANDARDS FOR COTS IN HIGH-INTEGRITY SYSTEMS

Regulatory agencies are charged with the responsibility of establishing processes, procedures, and mechanisms that are used to gain a appropriate level of assurance software will perform as intended. Agencies that deal with in safety critical domains have recognized the risks associated with using previously developed software and have started to address the problem. However, their actions in this area have not been comprehensive or consistent

Using COTS software in a safety critical application has recognized problems. Many agencies point out the issues by few identify prescriptive actions that can be taken. The Joint Services Computer Resources Management Group points out some of the issues associated with using COTS in critical applications.

> *"The safety assessment of Commercial Off the Shelf (COTS) software poses one of the greatest challenges to the safety certification of systems. COTS software is generally developed for a wide range of applications in the commercial market. The software is developed to an internal company standard or to an industry standard, such as IEEE, ANSI, or NIST. In general, the language used is determined by the company or the individual project team. Since the vendor releases only compiled versions of the product, there is often no way to determine which language is used. Because the developer can only guess at the applications that the software may be used in, specific issues related to application are often not addressed during the design." [8]*

NASA's Software Assurance Technology Center (SATC) recognizes the problem of data availability when dealing with previously developed software. They leave the decision up to the engineering judgement of the acquirer.

> *"For systems where use of this standard is required, it shall be applied to government furnished software, purchased software (including commercial-off-the-shelf (COTS) software), and any other reused software in the system. In the event that some of the analyses required by this document are not feasible due to the nature of the software and documentation, the developer is responsible for securing a waiver from the NASA acquirer of the system." [3]*

The Federal Aviation Administration calls out RTCA's Software Consideration in Airborne Systems and Equipment Certification (DO-178B). This is one of the most prescriptive software standards that are currently. This standard requires that all software, regardless of whether it was previously developed or not, meet the same high standards. It does not specify any procedures, or mechanisms that are to be specifically used for previously developed software.

> *"COTS software included in airborne systems or equipment should satisfy the objectives of this document. If deficiencies exist in the software life cycle data of COTS software, the data should be augmented to satisfy the objectives of this document" [6]*

The Nuclear Regulatory Commission and Lawrence Livermore National Laboratories have conducted extensive investigations regarding the design and development of safety critical software. The have also studied the use of previously developed software in High Consequence System Systems. They point out the fact that the state-of-the-practice in software engineering doesn't well support the use of previously developed software in systems where there is a high consequence of failure, but they leave open its use in systems with lesser consequence of failure.

> *"Given the current state of the art in software engineering and the technical, political, and maintenance considerations associated with typical commercial software, it appears that it will be difficult at best to incorporate COTS software products into high-consequence safety systems. If an effective grading process is in place, however, the tradeoffs associated with the use of COTS products in lower risk categories become more palatable." [7]*

In addition, they recognize the important part the vendor plays in the decision to use or not use previously developed software. It should also be noted that their comment specifically mentions programmable logic controller vendors. In this domain the software is relatively simple. Their comment might be less appropriate if, for example, the domain was software for knowledge based systems.

> *"Is it Possible to Use COTS Software in an HCSS? Yes. Some software vendors, such as programmable logic controller vendors, produce software with the knowledge that it will be used in systems with medium to high risks. Some of these vendors use software processes that have been designed to produce high-integrity software. They are generally aware of the types of hazards associated with the systems in which their products will be used and those hazards have been considered in their designs. In order to meet the demands of the high-integrity marketplace, they may be motivated to form long-term partnerships with users and to supply additional reliability documentation. Such vendors may well be in a position to meet applicable acceptance and regulatory requirements for the use of their products in HCSS's." [7]*

In traditional software developments, assurance that the software will function as intended is built up from several sources of evidence. These sources can broadly be classified as people and process, analysis, and testing [4]. There is agreement in the community that all three of these sources of evidence are necessary for true assurance. To date, the systems of interest to software assurance researchers have been primarily safety-critical systems. Typically, such systems are built from scratch with few, if any, COTS components. As discussed above, the regulatory community has not embraced the use of COTS software in safety critical systems. In addition, there are still significant questions about the cost-of COTS when used in safety critical systems [9]. However, there is increasing interest in expanding the use of COTS software components in critical, if not safety critical, systems.

3 NEW CHALLENGES WITH COMPONENT BASED SOFTWARE

Assurance of component-based software is complicated by the fact that its development and use differs from traditional COTS based software (e.g., libraries, operating systems, databases, and window management systems). In general traditional COTS software is configured for a specific operating system / processor pair, is used in somewhat predictable ways, and has a large user base. Component-based software introduces new assurance challenges that include:

- **Changeable environment and dependencies**– traditional COTS software is developed for a specific hardware and operating system specific environment. Vendors often support multiple hardware and operating system pairs, but the vendor knows these at development time. Software components execute in more abstract containers where the environment and the dependencies are not completely determinable at development time (e.g., dynamic class loading and model code). It should be noted that some major software "disasters" have been the result of changing environments [1, 2]. Because component-based software facilitates change it is likely to exacerbate this problem. Techniques and tools that assist in component dependency analysis are needed to improve this situation.
- **Variable usage** – one of the selling points of software components is their ability to be reconfigured for a variety of uses and to undergo post development configuration. From a functional perspective this is an attractive advantage, but it complicates assurance. For example, as a component's configuration changes it becomes difficult to extrapolate prior experience and past testing evidence. Also, for a given system, a component might have multiple configurations. The identification of the configuration of each component instance and a comparison with its assurance arguments becomes a post development assurance activity. Techniques and tools that assist in the extraction of component configuration descriptions are needed to improve this situation.
- **Larger number of smaller parts** – traditional systems have utilized a small number of rather large, well-defined COTS pieces. While the goal of component-based systems will be to use a small number of well-defined components to produce the desired functionality, they can potentially contain a larger number of less well-defined pieces. Techniques and tools that assist in the identification of component aggregation sets are needed to improve this situation.
- **Component flux** – is a problem similar to the version creep issue associated with the use of COTS software [9]. That is, the introduction of new versions is out of your control of the organization using the software. The problem is magnified with component-based software because of the increased number of components and connections between them. As the number of software components in a system increases the issue of identifying a consistent set of becomes more difficult. Techniques and tools that assist in the definition of component change analysis are needed to improve this situation.
- **Developer inconsistencies** – similar to traditional COTS software, by the time a software component is used in a system the group the developed it might no longer be together. This is complicated further when multiple development organizations are supplying components for the same system. Also, at present there are no authorities that certify individuals or organizations for the production of critical software components. While there are steps being taken in this direction [11, 12], meaningful advances are a decade away. This makes assessing the people and process difficult or impossible. Assurance is therefore best gained through analysis and testing.

Many of the challenges of using component-based software in critical applications can not be effectively addressed via traditional software assurance technologies. Testing approaches have been used for many years to show that software will function as intended. Given that components might not be developed by the organization using the component, white-box testing is generally not possible. Also, component testing is less effective when a component's configuration, environment, and dependencies can be changed at integration or deployment time. Performing component-level block-box testing can increase assurance, but, as pointed out in [12], developing the associated test oracles can be an expensive proposition. System and subsystem level testing can be effective, but they have a high associated cost, and can not cover the entire state space for most software. This means we must gain additional confidence through other means.

Process and personnel assessment is less useful when multiple organizations are involved and when components are externally developed (whether commercially or otherwise). New, more comprehensive assurance approaches are needed. These approaches might include the certification of software engineers and organizations, improved testing strategies, more fault tolerant designs, and better analysis techniques.

At the same time the assurance picture is getting more complicated, there are several factors are falling into place that will better enable the development of better assurance technologies for component-based software. Many of these factors are related to the use of the Java programming language and its associated Java Beans and Enterprise Java Beans technologies to developed highly reusable software components. These technologies are of particular importance because they are codified in commercial standards and support platform independence. Of interest to us are the potential improvements that can be made in the static and dynamic software analysis of component-based systems.

4 STATIC ANALYSIS FOR ASSURANCE OF COMPONENT-BASED SOFTWARE

Improvements in program analysis technologies, both static and dynamic, can help to overcome the challenges introduced by component-based software development. Using static component analysis that extracts and records relevant information from previously developed components is a tractable approach for improving assurance and offers some advantages over dynamic analysis. For example:

- Static dependency and exception propagation analysis can identify platform sensitive components.
- Characterizing parameter dependent behavior can identify components that are configuration sensitive.
- Combining information extraction with change analysis is a useful approach for identifying potential version incompatibilities.
- Identifying component sets with high cohesion and minimum coupling can form the bases for subsystem integration like testing in component-based software.

Our approach is not to invent new analysis techniques, as we believe there are many current analysis techniques that can be exploited. Rather, we want to investigate extending the currently available techniques and tools to adapt them to our purposes.

We are investigating the following areas:

- **Component descriptions –** methods and techniques to extract information from components that can be used to make static or dynamic assurance arguments. For example, inter-method dependencies, algorithmic configuration dependencies, exception propagation, and time dependent behavior. For those constraints that are particularly difficult to analyze, we are identifying ways to record and package off-line supplemental data in order to enhance static analysis possibilities or to support runtime based assurance techniques.
- **Component change analysis –** methods and techniques to identify the significant semantic differences among substitutable components. For example, changes in exception handling, use of threads, and use of internal data structure.
- **Component dependencies analysis –** methods and techniques to identify the significant dependencies among components and between a component and its platform or environment. For example, interface usage, platform dependent features, interaction closures, and component – framework dependencies.
- **Component aggregation sets -** methods and techniques to identify sets of components that exhibit high cohesion and minimum coupling. For example, sets of user interface, business logic, and data access components that exhibit logical cohesion, but reside in different architectural tiers.

These techniques will all help to address the difficulties identified in the previous section. Updated component descriptions that characterize environment-dependent can help to address the problem of assurance in the presence of a changeable environment and variable usage. Improved change analysis techniques are needed to address issues associated with component flux. Dependency analyses are required alleviate difficulties with the larger number of smaller parts and to provide visibility into the dependencies on a changeable environment. Finally, the identification of component aggregation sets can be used in the development of metrics more suitable for component based software, enabling more effective identification and remediation of developer inconsistencies.

5 CONCLUSION

We are focusing on development of a component assurance taxonomy, as well as information extraction and analysis strategies. Our taxonomy will identify assurance properties that are unique or magnified in component-based software, and approaches supporting analysis of these properties. A first step is the identification of analysis techniques and approaches that can be used in the determination of these properties. We are also investigating static information extraction techniques that can be applied to component-based software, particularly those written in Java. Where applicable, these strategies will be codified in an analysis workbench specification, a prototype analysis workbench, and in criteria useful for the development, acquisition, and analysis of component-based software.

REFERENCES

1. ARIANE 5, Flight 501 Failure, Report by the Inquiry Board.
2. N. Leveson, and C. Turner, An investigation of the Therac-25 accidents, *IEEE Computer,* 26(7): 18-41, July 1993.
3. *NASA Software Safety Standard*, NASA-STD-8719.13A, National Aeronautics and Space Administration, September 1997.
4. D. Parnas, et al., Evaluation of Safety-Critical Software, *Communication of the ACM*, 6 June 1990, Volume 33, Number 6, pg. 636-648.
5. President's Information Technology Advisory Committee (PITAC) Interim Report to the President, August 1998.
6. RTCA/DO-178B, *Software Consideration in Airborne Systems and Equipment Certification*, RTCA, 1992.

7. J. A. Scott, G. G. Preckshot, J. M. Gallagher, Using Commercial-Off-the-Shelf (COTS) Software in High-Consequence Safety Systems, *Lawrence Livermore National Laboratory*, 1995.

8. *Software System Safety Handbook*, Joint Services Computer Resources Management Group, 1997.

9. N. Talbert, The Cost of COTS, *IEEE Computer*, June 1998.

10. *Trust in Cyberspace*, National Research Council, Committee on Information Systems Trustworthiness, National Academy Press, 1999.

11. Standard for Safety-Related Software, UL-1998, *Underwriters Laboratories Inc*., 1994.

12. J. Voas, Certifying Off-the-Shelf Software Components, *IEEE Computer*, June 1998.