Software Certification and Software Certificate Management Systems

(Position Paper)

Ewen Denney and Bernd Fischer

USRA/RIACS, NASA Ames Research Center, Moffett Field, CA 94035, USA
{edenney,fisch}@email.arc.nasa.gov

1 Introduction

Software certification demonstrates the reliability and safety of software systems in such a way that it can be checked by an independent authority with minimal trust in the techniques and tools used in the certification process itself. It builds on existing software assurance, validation, and verification techniques but introduces the notion of *explicit software certificates*, which contain all the information necessary for an independent assessment of the demonstrated properties. A *software certificate management system* (SCMS) provides a range of certification services. It maintains the links between different system artifacts (e.g., design documents, engineering data sets, or programs) and different varieties of certificates, checks the validity of certificates, provides access to explicit audit trails, enables browsing of certification histories, and enforces system-wide certification and release policies.

We believe that a customizable SCMS with support for automated re-certification of diverse artifacts should become an essential part of any effective development process. Its primary impact is to increase the reliability and safety of software systems by providing automation support for their audit. A SCMS can at any time provide current information about the certification status of each component in the system, check whether certificates have been audited, compute which certificates remain valid after a system modification, and automatically start an incremental re-certification.

We are particularly interested in the combination of software certification with automated code generation and formal verification methods. Here, our focus is on the related questions of how code generators can support the certification process, and how software certification can be used to demonstrate and increase the reliability of the code generation process.

2 Challenges

Building reliable software is a challenging task in itself, but there are several challenges specifically related to certification, e.g.,

– maintaining high reliability, especially when a combination of diverse development techniques is used,

– minimizing certification efforts, especially for product families and interconnected systems of systems,
– reducing certification and re-certification times,
– linking between artifacts and certificates, and
– providing useful information (e.g., estimates of certification efforts).

We claim that the solution to these challenges is an intelligent, automated, and highly customizable software certificate management system integrated into the development process.

3 Software Certification

Software certification comprises a wide range of formal, semi-formal, and informal assurance techniques, including formal verification of compliance with explicit safety policies, system simulation, testing, code reviews and human "sign offs", and even references to supporting literature. Consequently, the certificates can have different types, and the certification process requires different mechanisms. A SCMS must be able to support such different certificate types and certification mechanisms. In order to guarantee separation of concerns and thus achieve scalability, certification approaches need to concentrate on individual risk factors one at a time. Consequently, a SCMS must be able to combine different certificates for the same artifact to construct an overarching certificate and, ultimately, to provide a higher degree of confidence.

Certificates A certificate contains all information necessary for an independent assessment of the properties claimed for an artifact. Obviously, the exact nature of the certificates depends on the nature of the artifact, the property, and the claim. However, a SCMS needs a unified view of certificates. At its most abstract, a certificate thus has to represent the three entities involved in the certification process, (i) the artifact being certified, (ii) the property being asserted, and (iii) the certification authority.

Certifiable Artifacts Certifiable artifacts include not only the conventional software artifacts (e.g., product families, completed systems, individual components, or even code fragments) but also supporting non-software artifacts: requirements documents, system designs, component specifications, test plans, individual test cases, scientific and engineering data sets, and others.

In particular, the supporting evidence for one certificate can be considered as the artifact of another certificate. For example, if the correctness of a component is to be certified using traditional black-box testing, the test harness and the test scripts are supporting evidence for the certificate; at the same time, the test harness can itself be certified, e.g., by a code review, and is thus the artifact of another certificate.

Certificate Hierarchies As indicated in the example above, the certificates for an artifact are not an unstructured collection but exhibit some hierarchical structure. This structure is determined by two independent dimensions, (i) the system structure, and (ii) the certificate types.

The internal structure of a *system* is reflected in the certificate hierarchy. If a system is decomposed into a number of subsystems, and each subsystem is built from a number of components, then a certificate for the system depends directly on the certificates of

the subsystems and indirectly on the certificates of all involved components. A SCMS must be able to represent this structure, taking into account language-specific visibility rules like module and subsystem boundaries that can limit the propagation of changes.

The second dimension is given by the certificates themselves, or more precisely, by the certificate types. The validity of a certificate can also depend on certificates for the supporting evidence (as described above), or even the authority, e.g., when a code review can only be signed by a certified software engineer. This part of the certificate hierarchy reflects the internal structure and procedures of the *organization* developing the software. A SCMS can then use the certificate hierarchy for auditing and incremental re-certification, similar to the way the Unix make-tool uses explicit dependencies and rules for incremental re-compilation. The SCMS can determine which certificates need to be inspected, recomputed, or revalidated after an artifact or a certificate has been (or would be) modified.

Certifiable Properties and Certification Authorities Traditional V&V has only addressed a restricted range of formal properties. Realistically, however, software development requires a wide range of notions of software reliability, safety, and validity, each with an appropriate certification authority. This must all be supported by a customizable SCMS. Examples include coding standards, test cases, statistical validity for data sets, simulation on high-fidelity test beds, fault tree analysis (FTA), failure modes and effects analysis (FMEA), stress tests, interoperability, usability, compatibility, and feasibility studies, as well as formally specified logical safety properties.

Release Policies In the context of certification, a release refers to the transition of an artifact into a new defined state: for example, launch, system integration test, alpha and beta testing phases, spiral anchor-point milestones, or code inspection. A release policy formally describes under which conditions an artifact is deemed to be in an adequately certified state and can thus be released safely to another state. Different release policies can be formulated to describe the different types of releases, and the corresponding certification requirements.

4 Certification Services

Intuitively, a SCMS combines the functionalities of a database (e.g., storing and retrieving certificates) and a make-tool (e.g., incremental re-certification). Specifically, it provides a variety of different services.

Certificate construction The main task of the SCMS is the construction of certificates. Given an artifact, a claimed property, and a certification authority, the SCMS will attempt to construct the certificate, invoking automatic mechanisms and notifying individuals of pending tasks, as appropriate. It should estimate the time and effort that the certification will take.

Editing and revoking Users can deem an individual certification authority to no longer be valid (e.g., a bug is discovered in a test harness, or an employee's badge has expired). The SCMS should revoke all certificates which depend on this.

Certificate maintenance The SCMS will carry out intelligent re-certification when a (customizably) appropriate change has taken place in the code or, more generally,

software artifacts to be certified. Existing (sub-) certificates should be reused where possible, especially where product families are concerned.

Auditing Since the SCMS provides a complete certification history with full information about all procedures followed, comprehensive audits can be carried out, applying alternative tools and/or oversight to any elements. The audit itself can then be recorded in the certification database.

Schema management Clearly, the SCMS must be generic. It must be customizable to existing procedures. It can be thought of as having a client-server architecture. The SCMS is the client and allows users to "plug and play" with arbitrary certificate servers.

5 Current Technology and Advances Required

In terms of computing infrastructure, the notions of certificate and certification are usually used in the context of security mechanisms for computer systems, in particular for computer networks and network-based services.

A SCMS can build on an existing secure infrastructure, e.g., PKI, for distribution, authentication, tamper-proof access control, persistence, and other desirable properties. However, there are a number of differences from existing technology where advances are required:

- linking to (and deep into) software artifacts,
- the wide diversity of forms of certification, both formal and informal, and
- the need for customizability and extensibility.

The SCMS should be an integral part of a development tool suite and use the same underlying datastructures, e.g., development graphs [1]. It can be linked to other tools, e.g., code generators and software reliability estimators. In particular, software certification can be combined with automated design documentation, so that the SCMS can provide an integrated exploration tool for code, certificates, and documentation, similar to safety cases [5] but more specific to the code level. Likewise, model-based software development tools should allow the definition of arbitrary domain-specific certificate types with respect to explicit domain models.

6 Certification of Automatically Generated Code

We are currently investigating some of these ideas in the context of an ongoing project on automated code generation. We have developed an approach to safety verification [3] in which the code generator is extended to enable Hoare-style safety proofs for each individual generated program. The key idea is to generate logical annotations along with the code, so that the proofs can be automated. These proofs ensure that the generated code does not violate certain conditions during its execution. However, it has gradually become clear that since this process produces a large number of auxiliary artifacts, and involves many components of varying complexity and reliability, that additional tool support should enable users to browse the entire set of safety artifacts.

In [4], we describe a rudimentary *certification browser*, which provides linking between the generated program, its verification conditions, the generated axioms, the proofs, and the proof checks. This is a first step towards an interactive tool which would, for example, allow designated users to sign off on otherwise unverified lines of code. This would be a prototype SCMS for code generation. Similar ideas have been investigated by the Programatica project [6], though not in connection with code generation.

In recent work we have developed an approach to inferring annotations for code produced by third-party code generators. This suggests a means of certifying the code produced by COTS code generators, and circumvents the difficulties that stem from treating these tools as black boxes.

7 Conclusions

Incremental certification and re-certification of code as it is developed and modified is a prerequisite for applying modern, evolutionary development processes, which are especially relevant for NASA. For example, the Columbia Accident Investigation Board (CAIB) report [2] concluded there is "the need for improved and uniform statistical sampling, audit, and certification processes". Also, re-certification time has been a limiting factor in making changes to Space Shuttle code close to launch time. This is likely to be an even bigger problem with the rapid turnaround required in developing NASA's replacement for the Space Shuttle, the Crew Exploration Vehicle (CEV). Hence, intelligent development processes are needed which place certification at the center of development. If certification tools provide useful information, such as estimated time and effort, they are more likely to be adopted. The ultimate impact of such a tool will be reduced effort and increased reliability.

References

[1] S. Autexier, D. Hutter, T. Mossakowski, and A. Schairer. The Development Graph Manager MAYA (System Description). In *Proc. 9th International Conference on Algebraic Methodology And Software Technology (AMAST'02)*. LNCS 2422, pp. 495–501, 2002.

[2] Columbia Accident Investigation Board Report, Volume 1. `http://caib.nasa.gov/`. 2003.

[3] E. Denney and B. Fischer. Formal Safety Certification of Aerospace Software. In *Proc. Infotech@Aerospace*. AIAA, 2005. Invited talk.

[4] E. Denney and B. Fischer. A Program Certification Assistant Based on Fully Automated Theorem Provers. In *Proc. International Workshop on User Interfaces for Theorem Provers, (UITP'05)*, 2005.

[5] T. Kelly and R. Weaver. The Goal Structuring Notation – a Safety Argument Notation. In *Proc. DSN Workshop on Assurance Cases: Best Practices, Possible Obstacles, and Future Opportunities*, 2004.

[6] Programatica Project. `www.cse.ogi.edu/PacSoft/projects/programatica`. 2004.