lagging enterprises. "Leaders," of course, are those companies with the most complete tool suites, while "laggards" are enterprises with minimal tool suites.

The table on the facing page is a first pass at a census of software tools and the ranges observed in the number of function points used. In round numbers, a fully equipped software engineering staff needs about 50,000 function points per staff member. For companies whose software staff supports multiple platforms (DOS, Windows, OS/2, Unix, MVS, VMS, and so forth), the needs will be even greater because of the current requirement to duplicate at least some tools for each environment.

To give some examples, Windows-based spreadsheets or word processors are in the 1,000 to 2,000 function-point range. Software cost-estimating tools range from fewer than 300 functions, for rudimentary tools that lack sizing or quality estimates, up to more than 1,500 function points for those with full sizing, logic, activity-based costing, and quality estimation.

When a CASE vendor advertises "full life-cycle support" but the product contains fewer than 10,000 function points (as many do), it's obvious that some important functions might be missing. This same approach can identify the missing components, which often are those associated with quality and configuration control.

Because the use of function points is new and still somewhat experimental, the results to date are interesting but not definitive. Even so, a method that takes some of the hot air out of vendor claims is worth exploring.

A method that takes some of the hot air out of vendor claims is worth exploring.

Structural models and patterned architectures

William Pollak and Michael Rissman
Carnegie Mellon University Software Engineering Institute

In 1986, the Aeronautical Systems Command (ASC) Program Office for Simulators and Training Devices began using software architecture as a systems engineering technology under the technical leadership of Bill Schelker. The purpose of this initiative was to reduce the technical risk inherent in developing high-quality training simulators for advanced aircraft.

Flight simulators provide safe, cost-efficient training for aircrews. Digital computer technology has been used to reproduce the behavior of aircraft for more than 25 years. However, as aircraft and training missions became more complex, the traditional design for training devices was stretched to its limit. Traditional designs, optimized for efficiency, could not contend with the complexity of modern aircraft and training missions. Despite the ingenuity of simulation software engineers, simulation systems were difficult to integrate, and once integrated they were difficult to modify as the simulated aircraft and its mission evolved.

The new architectural approach dealt with the complexity of modern training devices by capturing patterns of organization in a simple structural model. This structural model exposed deep levels of communication and coordination within and across components, subsystems, and systems. The fully analyzed structural model was used consistently and predictably to specify and implement the required simulation models. The resulting architecture manifested multiple instances of the pattern that the structural model embodied.

With major contributions from simulator contractors and the Software Engineering Institute (SEI) at Carnegie Mellon University, and with ongoing support from ASC, the original structural models have evolved and have been used successfully on a number of recent acquisitions.

Structural models. A structural model is a collection of structural types manifesting a particular pattern of interaction. Logically, each element of a structural model is a placeholder for problem domain functionality. Physically, each element provides the computing resources needed to execute that kind of functionality. Thus, a structural model is a straightforward ontological mapping of problem space entities to solution space entities.

Structural models resolve architectural issues common to multiple subsystems. For example, a flight simulator models the functionality of many aircraft subsystems. The subsystems pose a common set of architectural challenges, including the partitioning of the subsystem into components, propagation of state among components, importing and exporting of state for other subsystems, and contention for shared hardware resources.

Structural models consist of a small number of structural elements manifesting clear patterns of interaction. The architectures of systems developed using structural models are characterized by multiple instances of ensembles that are composed from these structural model elements. Thus, the complex interactions among the system's many architectural units emerge from simple
patterns of interaction among this small number of elements.

A structural model embodies systems engineering decisions about common architectural issues. Proper use of the structural model assures that the system will manifest the qualities that the model embodies. A structural model is created by a systems engineering team early in the development process. The team evaluates the structural model in terms of both required system functionality and desired system qualities. The structural model is then represented as a set of specification forms and code templates. The specification forms are used to describe instances of the elements; the code templates are used to package the implementations of the specified instances.

A top-level allocation of functionality to instances of the structural model is used to create an integration harness. Estimates of runtime resources required by the instances are used to generate synthetic loads on the runtime resources. Development proceeds in the context of the synthetically loaded integration harness. As development proceeds, synthetic loads are refined and ultimately are replaced by actual implementations of the specified functionality.

Benefits of using structural models. The ability to integrate and modify software depends on its overall structure. By making the deep structure explicit at the earliest possible stage in the design process, structural models allow systems engineers to evaluate the software structure with respect to qualities like integrability and modifiability. Since a structural model reduces the organization of a complex system to interactions among a small set of structural types, structural models allow systems engineers to comprehend the system trade-offs that affect such important qualities.

Structural models also provide a discipline for coordinating development efforts across a large project. The simulator is designed and implemented in the context of the skeletal architecture. This architecture provides a workspace for designers of individual subsystems. Thus, each subsystem is developed within the system of which it is a part, and integration problems are significantly reduced. Furthermore, the structural model makes the intent of the systems engineers clear to the engineers specifying and implementing the various simulation models, and it provides a basis for organizing all information about the system under development.

Research issues. Structural models have been used primarily on a class of real-time, man-in-the-loop simulators referred to as aircrew training systems. They have been used extensively on acquisitions such as the B-2 Aircrew Training Device, the C-17 Aircrew Training System, the Special Operations Forces family of trainers, and a number of other simulators acquired by the Air Force and by other branches of the US military.

To date, however, no program has used structural models to their full potential as a systems engineering technology. Full use will eventually require changes to acquisition practices. In addition to defining best acquisition and development practices involving structural models and patterned architectures as a systems engineering technology, the Air Force intends to involve the simulator community in a number of other activities: improving current structural models for simulators, creating structural models for domains of simulator functionality not addressed by existing structural models, developing tools that automate the use of structural models, and using structural models to improve estimation and tracking of life-cycle costs.

At the SEI, work on structural models is part of a broad initiative focusing on the role of software architectures and associated technologies. The general goals of this initiative are to define the role of architecture in computer-based systems development, to identify current architecture technologies, and to coordinate the activities of various communities developing and using such technologies.

Acknowledgment

This work is sponsored by the US Department of Defense.

William Pollak is a senior editor at the Software Engineering Institute, Carnegie Mellon University.

Michael Risman is a member of the technical staff at the Software Engineering Institute, Carnegie Mellon University.

Readers can contact the authors at the Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213-3890; e-mail [wp, mmr]@sei.cmu.edu.

Formal methods

Jonathan Bowen, Oxford University, and Michael G. Hinchey, University of Cambridge

In recent years, the television news and the popular science journal segments of the media have become preoccupied with the failures of safety-critical computer systems. A number of systems or classes of system have particularly caught the media's attention: nuclear power plants where cooling systems or shutdown loops have demonstrated inconsistency and air crashes that could not be blamed on pilot error.

The introduction of computer systems to replace more traditional mechanical systems (for example, Boeing’s fly-by-wire system in the 777 jet) has raised the awareness of the system-development community and the general public to perceive the unprecedented opportunities for the introduction of errors that computers permit.

Many journalists have become self-styled authorities on techniques that increase confidence in the correctness of such systems and reduce the number and frequency of computer errors. Formal methods constitute one of the techniques that have been widely advocated for this purpose, and their usage is being suggested in an increasing number of standards in the safety-critical domain. Here, we compare the recommendations given in a number of important existing and emerging standards, and attempt to identify future trends in this important area.

Formal methods. The level of industrial usage of formal methods in safety-critical and security-critical areas continues to grow rapidly, although their usage is still more the exception than the rule.

There are many misconceptions as to the exact nature of formal methods and what they can actually do. Consider, for example, two alternative definitions of
and safety-critical standards

formal specification taken from an IEEE glossary:

(1) A specification written and approved in accordance with established standards.

(2) A specification written in a formal notation, often for use in proof of correctness.

The latter is the sense assumed by those involved in the development and exploitation of formal methods, but the former may be more prevalent in industrial circles.

It is now widely accepted, however, that formal methods have potential benefits that are likely to be exploited increasingly in the fields of safety- and security-critical systems. A number of standards, particularly in the safety-critical domain, are now citing formal methods as one of the techniques that should be employed when the highest integrity of software is required.

Governmental legislation is likely to further encourage the use of formal methods and to prompt their inclusion in various emerging standards. For example, the Machine Safety Directive issued by the European Commission in effect since January 1, 1993, allows for claims under civil law against the supplier of machinery whose logic (including its software) is determined to be in error. Criminal proceedings may be taken against the responsible parties, should negligence during design or manufacture be proven. Suppliers must demonstrate that they have employed best practice in the development process; this is increasingly likely to encompass the use of formal methods.

Standards. Until recently, few standards have been concerned specifically with software in safety-critical systems. Often, software quality standards such as the ISO 9000 series of the International Standards Organization have been used instead, since they were the nearest relevant guidelines. We believe that many emerging standards will strongly recommend formal methods, or indeed make their usage mandatory in certain classes of applications.

A number of standards bodies have made this move, although some have done so at a superficial level; surprisingly, others fail to mention formal methods at all. Table 1 summarizes the survey of standards covered.

IEEE P1228. The appendix of an early draft report by the IEEE Computer Society Software Engineering Standards Subcommittee Safety Plans Working Group (P1228) included headings on “Formal/Informal Proof” and “Mathematical Specification and Verification” as techniques under consideration for inclusion. More recent drafts of “Standard for Software Safety Plans” omit mention of formal methods. As a “plan” standard, it avoids mentioning specific techniques. However, it is likely to be an important standard in the safety-critical area, and formal methods may be applied in projects adhering to this standard in the future. P1228 has been accepted as a standard, available from the IEEE since June.

US DoD Mil-Std-882. Not surprisingly, US Department of Defense Mil-Std-882B, “Military Standard: System Safety Program Requirements” (1984) made no reference to formal methods, although it did mention a number of specific techniques. Mil-Std-882C (1993) was expected to mention formal methods but didn’t. Indeed, unlike B, revision C makes no specific reference to techniques. The standards are divided into tasks for program management and control, design and integration, design evaluation, and compliance and verification, but no software specific tasks.


This subsection introduces formal methods and describes three levels of rigor: (1) formal specification without proofs, (2) formal specification with manual proofs, and (3) formal specification with automatically generated or checked proofs.

Manufacturers following the guidelines may use formal methods in the context of aircraft certification, although the responsibility of justifying usage still lies with the manufacturer. This weak RTCA endorsement is extremely disappointing considering the significant amount of lobbying by the aviation industry, system development specialists, and the media for more rigor in the development of aircraft systems.

AECB. The Atomic Energy Control Board in Canada commissioned a proposed standard for computer software in nuclear power station safety systems. The report, originally based on the IEC Standard 880 (“Software for Computers in the Safety Systems of Nuclear Power Stations”) and prepared by David L. Parson of McMaster University, has been revised substantially, placing a much greater emphasis on documentation.

The report is not a standard itself, but rather is used as a form of “meta-standard” — a standard for evaluating standards and procedures submitted by licensees, many of which (such as those by Ontario-Hydro) are among the few standards to specifically mandate formal methods.

IEC. The International Electrotechnical Commission has updated its two
standards in the area of safety-critical systems. These standards are intended to be generic but applied in different industrial sectors as particular instantiations. “Software for Computers in the Application of Industrial Safety Related Systems” (IEC65-Working Group 9) and “Functional Safety of Programmable Electronic Systems: Generic Aspects” (IEC65-WGlO) explicitly mention the “formal methods” CCS, CSP, HOL, LOTOS, OBJ, Temporal Logic, VDM, and Z. A short section on formal proof is also included in the former.

There are plans for three IEC standards on “Functional Safety of Programmable Electronic Systems”: (1) generic requirements, (2) requirements for electrical/electronic/programmable systems, and (3) software requirements.

**Table 1. Software-related standards and guidelines.**

<table>
<thead>
<tr>
<th>Region</th>
<th>Body</th>
<th>Sector</th>
<th>Name</th>
<th>FMs Content</th>
<th>FMs Mandated</th>
<th>Status</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>US</td>
<td>IEEE</td>
<td>Generic</td>
<td>P1228</td>
<td>No</td>
<td>n/a</td>
<td>Standard</td>
<td>1994</td>
</tr>
<tr>
<td>US</td>
<td>Department of Defense</td>
<td>Defense</td>
<td>Mil-Std-882B</td>
<td>No</td>
<td>n/a</td>
<td>Standard</td>
<td>1985</td>
</tr>
<tr>
<td>US</td>
<td>DoD</td>
<td>Defense</td>
<td>Mil-Std-882C</td>
<td>No</td>
<td>n/a</td>
<td>Standard</td>
<td>1993</td>
</tr>
<tr>
<td>US</td>
<td>Requirements and Technical Concepts for Aviation</td>
<td>Aviation</td>
<td>DO-178A</td>
<td>No</td>
<td>n/a</td>
<td>Guideline</td>
<td>1985</td>
</tr>
<tr>
<td>US</td>
<td>RTCA</td>
<td>Aviation</td>
<td>DO-178B</td>
<td>Yes</td>
<td>No</td>
<td>Guideline</td>
<td>1992</td>
</tr>
<tr>
<td>Canada</td>
<td>Atomic Energy Control Board</td>
<td>Nuclear</td>
<td>2.234.1</td>
<td>Yes</td>
<td>Yes</td>
<td>Report</td>
<td>1992</td>
</tr>
<tr>
<td>Canada</td>
<td>Ontario Hydro</td>
<td>Nuclear</td>
<td>982C-H69002</td>
<td>Yes</td>
<td>Yes</td>
<td>Procedure</td>
<td>1993</td>
</tr>
<tr>
<td>Europe</td>
<td>European Space Agency</td>
<td>Space</td>
<td>PSS-05-0</td>
<td>Yes</td>
<td>No</td>
<td>Standard</td>
<td>1991</td>
</tr>
<tr>
<td>Europe</td>
<td>International Electrotechnical Commission</td>
<td>Generic</td>
<td>IEC65AWG9</td>
<td>Yes</td>
<td>No</td>
<td>Draft</td>
<td>1989</td>
</tr>
<tr>
<td>Europe</td>
<td>IEC</td>
<td>Generic</td>
<td>IEC65A 122</td>
<td>Yes</td>
<td>No</td>
<td>Proposed</td>
<td>1991</td>
</tr>
<tr>
<td>Europe</td>
<td>CENELEC</td>
<td>Railway</td>
<td>SC9XA/WGA1</td>
<td>Yes</td>
<td>No</td>
<td>Draft</td>
<td>1993</td>
</tr>
<tr>
<td>UK</td>
<td>Ministry of Defence</td>
<td>Defense</td>
<td>00-55</td>
<td>Yes</td>
<td>Yes</td>
<td>Interim</td>
<td>1991</td>
</tr>
</tbody>
</table>

The document also advocates formal proof of correctness wherever practicable, and suggests that proofs should be checked independently to reduce the possibility of human error. Although they are not specifically mandated, formal methods are highly recommended, and formal proof is recommended before any testing is undertaken.

CENELEC. In Europe, CENELEC (an abbreviation for the European Committee for Electrotechnical Standardization), concerned with railway safety, has produced a draft “European
Norm” document on “Safety-Related Software for Railway Communications, Signaling, and Processing Systems” for comment. The document is designed to be read in conjunction with the IEC 65A 123 standard. Formal methods are highly recommended for the highest of four safety integrity levels and recommended for the middle two levels in the areas of software requirements specification, design and development, and verification using formal proof. CCS, CSP, HOL, LOTOS, OBJ, Temporal Logic, VDM, and Z are explicitly mentioned. “Highly recommended” means that if the technique is not used, then the rationale behind this must be detailed in a quality plan and agreed with the assessor. “Recommended” means the technique may be combined to form part of a package of techniques. Appendix B gives a list of possible techniques, including specific formal methods with references.

UK MoD 00-55 and 00-56. The UK Ministry of Defence (MoD) provides a major impetus for the exploitation of formal methods by various regulatory bodies. Interim Defence Standard 00-55 (“The Procurement of Safety Critical Software in Defense Equipment,” 1991) is divided into two parts: Part 1 deals with requirements, and Part 2 offers guidance on this matter. The 00-56 standard concerns hazard analysis and safety classification of the computer and programmable electronic system elements of defense equipment. Like 00-55, it is divided into two parts concerning requirements and guidance. Annex C of part 2 provides a Z specification of some of the tables in part 1. Many of the procedural elements of the revised standard have also been formally specified by Formal Systems (Europe) Ltd., again using the Z specification language. This has not yet been incorporated into the standard itself.

These interim standards have caused a degree of controversy and much discussion in the UK system-development community and defense establishment due to the extent to which they (particularly 00-55) mention and mandate formal methods.

The 00-55 standard mandates the production of specifications of safety-critical modules in a formal language. These must be analyzed for consistency and completeness with respect to all potentially hazardous data and control flow. Furthermore, all safety-critical software must be subjected to validation and verification to establish that it complies with its formal specification. This includes static and dynamic analysis as well as formal and informal (but rigorous) proof.

Discussion. Current trends in standardization and the number of standards in the safety-critical domain that are under discussion augurs many more future standards in this area. Many are likely to be industry-specific standards (such as for the railway, avionics, and nuclear industries), based on more generic standards such as those proposed by the IEC. Thus, recommendations in international generic standards are likely to have significant influence in the long term.

Many of these standards are mentioning formal methods, and others are likely to lean in that direction. It should be noted, however, that most standards bodies are recommending formal methods rather than mandating them; the UK Ministry of Defence and the Canadian AECEB are exceptions. It is disconcerting that there is no mention of formal methods in the latest version of the DoD Mil-Std-882, but this perhaps reflects market resistance to the acceptance of formal methods even in safety-critical systems. This is often due to ignorance and misconceptions regarding the costs, benefits, and difficulties of using such methods, many of which are unfounded.4,5

Formal methods and notations are themselves becoming standardized. There is a standard for LOTOS (ISO 8807, issued in 1989), a formal description technique (FDT) based on the temporal ordering of observational behavior. VDM-SL and Z, two of the leading general-purpose formal specification languages, are undergoing international standardization under ISO/IEC Joint Technical Committee 1/Subcommittee 22. The X3J21 technical committee on FDTs, under the X3 Accredited Standards Committee on Information Processing Systems, is overseeing these standardizations and is interested in future developments such as object-oriented extensions to FDTs and their possible standardization.

The inclusion, or recommendation, of formal methods in a number of standards as a means of attaining high integrity software reflects the importance and relative maturity of formal methods. We believe this trend will continue, and we expect that formal methods can help to improve safety standards and make them more unambiguous. The use of formal methods in standards will provide motivation for further research, funding, teaching, and exploitation of formal methods in software development.

We hope this will eventually lead to improvements in safety levels and help protect the lives and resources that are reliant on complex computer software.

Acknowledgments

Besides those mentioned in Reference No. 1, we thank Kirsten M. Hansen, Bill Harvey, David Levan, and Tony Zawilski for their contributions.

References


Jonathan Bowen is a senior research officer at the Oxford University Computing Laboratory, Programming Research Group, Wolfson Bldg., Parks Rd., Oxford OX1 3QD, UK, phone 44 (865) 283512 (273838 general), fax 44 (865) 273839, e-mail jonathan.bowen@comlab.ox.ac.uk.

Michael G. Hinchey is a researcher with the University of Cambridge Computer Laboratory, New Museums Site, Pembroke St., Cambridge CB2 3QG, UK, phone 44 (223) 334419 (334600 general), fax 44 (223) 334678, e-mail michael.hinchey@cl.cam.ac.uk.