Where Do Software Security Assurance Tools Add Value?

David Jackson
QinetiQ
WWA109 Malvern Technology Centre
Malvern, WR14 3PS, UK
[+44] (0)1684 896689
DMJackson@QinetiQ.com

David Cooper
CESG
Room A2h, Hubble Road,
Cheltenham, GL51 0EX, UK
[+44] (0)1242 221491 ext 39049
David.Cooper@cesg.gsi.gov.uk

ABSTRACT
In developing security information technology products, we are presented with a wide choice of development and assurance processes, and of tools and techniques to support those processes. By considering a structured break-down of the goals of a development, and building on the results of a survey of the applicability of tools to certification, this paper proposes a framework for assessing the value of tools – both security specific and more general – to security assurance.

Categories and Subject Descriptors
D.2.9 [**Software Engineering**]: Management–software quality assurance (SQA); Software/Program Verification–Validation

General Terms
Management, Measurement, Security.

Keywords
Software Assurance; Common Criteria for Information Security Evaluation

1. INTRODUCTION
Security is important in all aspects of life, and the increasing pervasiveness and capability of information technology makes IT infrastructure security increasingly so [1]. The continual and increasing publicity given to failures of IT security demonstrate the importance of developing and assuring systems to appropriate levels of security.

In spite of this attention, security remains a difficult attribute to assess and value [2]. Although the benefits of improved security can be difficult to quantify, as technologists and managers we are required to define and implement security measures which are appropriate to the threat and to the application.In the area of software security, these choices are further complicated by the wide range of techniques and tools have been used or proposed. Efforts are being made to categorize these tools and techniques, and to measure the effectiveness with which they perform their functions, but the variety of different approaches makes direct comparisons difficult.

This paper is a preliminary attempt to identify the role of various assurance activities and tools in the development of a software product, and the potential benefits of employing them. We believe that virtually all developments aimed at a non-trivial distribution will require some degree of security assurance.

This paper is based on the authors' experience in a number of recent projects relating to software security assurance. Its principle inputs are:

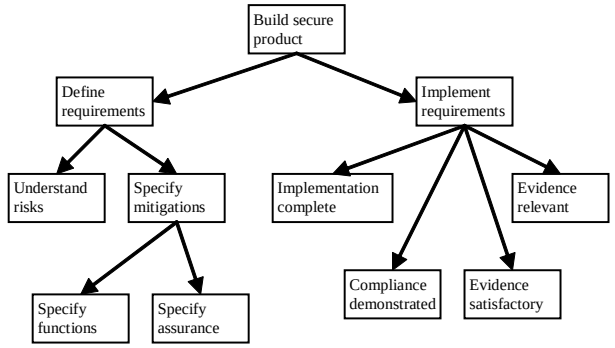
- A study carried out on behalf of the UK Government CESG into the use of tools in support of Common Criteria (CC) evaluation [4];
- The SafSec project, which is investigating cost-effective safety and security certification approaches for Integrated Modular Avionics (IMA) [5]; and
- Discussions around the NIST workshop on "Defining the State of the Art in Software Assurance Tools" [6].

The work described here is the first attempt to combine the goal-based approach proposed by SafSec with the survey results of the other projects, and also takes into account the recent revision of the Common Criteria [17]. As a result, it poses questions for future research which are more wide-ranging than earlier studies.

2. BACKGROUND – THE ASSURANCE PROBLEM
Various approaches are used by those responsible for developing, deploying and maintaining IT equipment and systems. Historically, most of the emphasis on information security came from government and military applications. Information security techniques developed which were appropriate for these highly-regulated environments. These are typified by formal product approval schemes such as that established by the Common Criteria for Information Technology Security Evaluation [3] (hereafter Common Criteria or CC). In purely commercial applications, less rigorous division will typically exist between development and security assessment, but effective security processes will still generally contain elements of both [7]. In order to examine where the benefits of particular technologies in supporting security assurance lie, we will consider a general model of product development, taken from [5].

2.1 Lifecycle of a Secure Product

Figure 1 High-level Goals of a Product Development



The security aspects of a development address a number of goals; these goals do not necessarily represent particular activities, but rather aspects of development which must be made, and maintained, valid through the course of development and deployment. A high-level view of a typical project is shown in Figure 1, which is based on that adopted by the SafSec project [5]. The notation is based on Kelly's Goal Structuring Notation (GSN) [8]. The key goals are grouped into those which derive security specifications (understanding the risks, specifying mitigations) and those which ensure the specifications are implemented (completion and control of implementation, generation and adequacy of assurance evidence).

Although the emphasis given to these goals will differ widely between products according to the priorities of particular industries and applications, at some level each of these areas must be addressed by an adequately assured development.

Given a breakdown of the goals which a product development is seeking to achieve, we can assess the value of project activities by considering their contribution to meeting particular goals. Ultimately, we might assess the relative merits of different strategies by considering the relative economy (in terms of the necessary supporting solutions) with which each supports the goal. Obviously if the goals have been characterized purely in terms of security, only security aspects of the development will be illuminated by such an analysis.

2.2 Common Criteria Evaluation & Practical Security

In regulated applications, these development goals are often satisfied by adopting a formal certification scheme, of which the Common Criteria are the most widely accepted. Certification schemes generally involve additional time and expense in meeting their requirements, and thus the value of such schemes has been questioned. Areas in which the results of a certification program may differ from expectations include:

- Measurement of results: Is the objective to minimize vulnerabilities discovered or published, or to achieve a level of confidence that no significant risks remain?
- The scope of assessment: some evaluations are carried out under constraints which are too stringent to be widely practicable.
- Development processes: development technologies are continually evolving, and future developments may not match the expectations of the certification scheme.

Our previous work [4] includes a study aimed at addressing some of these issues and reviewing both assurance technologies and the CC assurance criteria to identify potential improvements to development and evaluation processes. The baseline for the work described here is the current formal release of the common criteria, version 2.2. The implications of the new draft of the CC, version 3, are discussed in Section 4.4 below.

2.2.1 Current evaluation practice

Current evaluation practice is driven by the evaluation method [9]. Many requirements are focused on a product's documentation, rather than any formal artifact. This documentation – models of the design, or representations of the implementation, for example – is typically largely manually generated and intended for manual review. The CC evaluation process also makes assumptions about the development process. The information available is assumed to be consistent with a waterfall-style development: security functions are identified at the requirements level, and their presence and correct implementation verified through successive levels of design representation, culminating in their demonstration (by testing) in the final product. In consequence, only a small proportion of the evaluation effort is typically spent examining the product (code). Focus on implementation of identified security functions also poses pragmatic problems for evaluators: to make a sensible judgment on security issues, a thorough understanding of the product is necessary, but the targeted documentation provided to trace the security functions will not necessarily help build this understanding. Emphasis on the presence of specific security measures is also seen as encouraging the de-scoping of valuable, but difficult-to-assure, measures from the security targets which are claimed. This could result in the accreditation of products with increasingly unrealistic constraints on their operation, as opposed to real improvements in security. These factors do not encourage the types of assurance (e.g. scanning for potential vulnerabilities, or automated checks for compliance with implementation rules) that are amenable to automation (apart from standard document automation functions such as searching and indexing.). The product development process is, of course, likely to make some use of tools, for example to manage and organize source code, to generate and monitor tests, or to carry out customized consistency or style checks. Typically, however, such tools are used purely to benefit the development, not to contribute to the formal security assurance process. Evaluation may be facilitated by tools which contribute to the management of development (such as tools for configuration management, impact analysis, change control, or automated build and testing) but to no greater extent than any other process is facilitated by having good control of its inputs.

2.2.2 Desirable changes

The perceived weaknesses of current assurance regimes lead us to try and identify desirable features of future assurance approaches. Key attributes identified by a range of stakeholders in the CC scheme included:

- Assurance should not introduce significant delay or additional cost into product development.

- Emphasis should be placed on identifying vulnerabilities in the product, rather than exploring attributes of the documentation.
- Assurance requirements should not list specific documents and checks, but allow flexibility to choose development environments and life-cycles which reflect current practices, and the goals to be addressed.
- Encouragement of broader good practice and approaches that facilitate understanding and assurance of the whole product, not simply a set of security functions. Also promotion of security targets which are broad enough to be practically applicable, rather than restricted to facilitate accreditation.
- Provision of concrete advice on the application of specific techniques, the use of tools in specific areas, and the identification and elimination of particular high-risk structures.
- Maintenance of the existing high standards for the quality of assurance, eg by maintaining mutual recognition and repeatability, and demanding appropriate validation of tools.

However, it was emphasized that any changes should not jeopardize the assurance of systems which are not covered by available tools, or introduce unrealistic expectations of the developers (e.g. by demanding manual resolution of many false positive reports from tools).

3. CLASSIFICATION OF ASSURANCE TOOLS AND TECHNOLOGIES

The tool survey carried out as part of the CC investigation examined a broad range of technologies, on the grounds that many tools might contribute to developing a secure system even if they are not specifically security-related. The classification used there is summarized here.

- Tools which aid human comprehension of software, including
    - Reverse-engineering to graphical representations
    - Enhanced code navigation
    - Automatic documentation
    - Presentation of multiple views
    - Configuration management
    - Integrated development environments

  There are also tools targeted specifically at audit and assessment, which typically include a number of the above functions.
- Configuration management tools. Some form of configuration management is essential, but in this category we include related tools for controlling and supporting development. Additional facilities offered include:
    - Comprehensive documentation management (not just source code)
    - Change management
    - Version / variant management
    - Traceability
    - Build management
    - Access control
    - Integration with the development environment
- Test support and dynamic analysis tools, covering not only conventional testing, but also other assurance activities based on execution of (a variant of) the product. Examples include:
    - Test execution frameworks
    - Test case generation, both white-box (based on the implementation) and black-box (based on a separate specification of intended behavior)
    - Test coverage analysis
    - Memory and execution profiling
- Subset conformance tools. Some forms of security risk can be avoided by eliminating certain classes of structure from allowable implementations, essentially defining a subset of the implementation language. These subsets can be standardized (as, for example, the MISRA subset of C [10]) or company- or project-specific.
- Detection of general implementation weaknesses. Many means of exploiting vulnerabilities make use of errors in software implementation, even if the errors themselves do not constitute a direct vulnerability. Detection and elimination of general programming errors will improve the overall quality of a software product and reduce the potential for security functions to be bypassed or subverted.
- Run-time error detection. One specific class of software weakness which can be difficult to identify by testing is the occurrence of run-time exceptions such as overflow and arithmetic errors. Several approaches have been developed to identify where such errors may occur.
- Vulnerability detection. Of all the classes of flaws which we may search for in a product, those which present known vulnerabilities offers the most direct benefit. A range of tools is available according to the implementation technology and vulnerability classes of concern. This area is the main focus of many other surveys, including [11].
- Executable code analysis. Many of the attributes noted above can be determined either at source code level or by direct examination of object code or byte-code. Source code tools have potential access to richer information about the design intent that object-code tools, but the latter have the advantage of applying directly to the delivered product, and could, for example, be applied to third-party or legacy components.

- Program correctness tools. Although typically applicable only to higher levels of assurance requirement, and thus of limited general applicability, some tools do exist which address the question of program correctness in a broader sense. As many security vulnerabilities are likely to lie outside the intended behaviour of a program, these are able to provide high levels of confidence in the security of a product. Typically, however, they require additional design and implementation effort, such as the preparation of formal specifications or program annotations.

Another recent proposal for a taxonomy of security assurance tools [11] identified the following classes:

- External
    - Network Scanners
    - Web Application Scanners
    - Web Services Network Scanners
    - Dynamic Analysis Tools
- Internal
    - Software Requirements Verification Tools
    - Software Design/Modeling Verification Tools
    - Source Code Scanning Tools, further divided into identification of range and type errors, calls to vulnerable library functions, environmental problems synchronization and timing errors, locking problems, protocol errors, general logic errors and other flaws (file cycling issues, access control implementation, configuration violations)
    - Byte Code Scanning Tools
    - Binary Code Scanning Tools
    - Database Scanning Tools

This breakdown provides more detail on security-specific tools, and includes, in its external category, tools that, being most useful after deployment, were not judged relevant to a product assurance process for the purposes of our earlier study. It provides less detail on tools which are not security specific. Ongoing work to develop a more general taxonomy is taking place as part of the NIST SAMATE project [6].

If tools are to be used in creating or assessing assurance evidence, it is necessary for the tools themselves to be fit for the purpose, in order to establish the requisite confidence in the results they produce. The problem of tool qualification is not unique to security, and has been addressed, for example, by the aerospace safety community [12]. The benchmark for any tool which replaces a life-cycle process is that its output should be at least equivalent to the processes replaced; this means that if the output of a tool is cross-checked independently by some other activity, the requirements place on the tool itself may be relaxed.

Attributes which may be expected of a qualified tool include:

- Clear definition of the function performed and requirements satisfied
- Accuracy
- Repeatability
- Completeness and lack of ambiguity of output
- Characterization of operating environment and behavior under abnormal conditions
- Demonstration of requirements coverage, and analysis of the degree of coverage achieved
- Evidence of previous evaluations, of previous successful deployments, and of the pedigree of other tools developed by the same process
- A traceable defect recording and corrective action system

Ultimately, if the requirements on a class of tool can be characterized with sufficient accuracy, we could expect to develop certification criteria and independent testing schemes.

4. POTENTIAL BENEFITS

The CC classify security requirements into security functional requirements (specifying particular security-related functions which a system must provide) and assurance requirements (specifying the measures to be taken to ensure correct implementation of the functional requirements). Assurance requirements are further subdivided into families:

- Configuration management (ACM)
- Delivery and operation (ADO)
- Development (ADV)
- Guidance documentation (AGD)
- Life cycle support (ALC)
- Tests (ATE)
- Vulnerability assessment (AVA)

Seven pre-defined packages of assurance requirements are defined, representing increasing levels of assurance – the Evaluation Assurance Levels (EAL) 1–7 where EAL 1 is the least stringent, and EAL 7 the most.

Analysis of the capabilities of the various classes of software development and assurance tools against the CC requirements led us to consider three areas in which tools can facilitate the development of an assured product, as follows.

- Tools employed in the development, but which support or facilitate assurance,
- Tools of direct use in evaluating security, and
- Tools which support the implementation of security functional requirements rather than providing evidence that security assurance requirements have been met.

These areas are discussed in the following sub-sections.

One additional aspect of tool use became clear in the course of the analysis: there are many areas in which tools which may not necessarily assist in one-off assurance of a particular development but contribute substantially to the effective maintenance and re-use of assurance evidence. Such re-use is important in many situations:

- In re-assurance of a modified or updated product,
- In assurance of a product in distinct, but related, environments (eg across different platforms), and
- In developing composite systems that make use of previous assurance evidence about their components.

Of particular importance in these cases is the need to be able to identify where modifications have been made, and where dependencies arise which may need to be re-considered in the light of those changes. Our experience indicates that even given such facilities, re-assuring a complex system can still be difficult if an attempt is made to re-use parts of previous work in the production of complete new assurance arguments; re-use is more likely to be effective if complete assurance arguments are used as a whole, forming a baseline against which later assurance is documented as an assured set of changes.

4.1 Tools employed in development

Many tools used in development are useful in supporting assurance, because many of the factors which facilitate assurance also directly facilitate the development itself. Nevertheless, some development tool functions are of greater relevance than others. Areas of particular relevance are described below

4.1.1 Configuration Management (CC Assurance Class ACM).

Tool support for change and build management provides both developers and assessors with confidence that the product delivered – and its supporting configuration information, documentation, training material, etc – are derived from valid sources and controlled appropriately. All serious product developments will use some forms of configuration management policy and tools; nevertheless, choice of appropriate tools can greatly simplify assuring an appropriate level of configuration management. Features of particular relevance include: comprehensive coverage of all documents (not simply code); access control; change control; traceability; and version comparison/impact analysis to support re-use of assurance evidence.

4.1.2 Life cycle support (CC Assurance Class ALC)

Confidence in the control of the development life-cycle is an important component of assurance. While few tools control the life-cycle directly, and lifecycle definition and control are general project-management issues beyond the scope of this paper, a number of aspects of assurance benefit from an appropriate development tool environment. Configuration management tools which provide formal release control, for example, may be used to enforce compliance to particular life-cycle features. Development security (Class ALC_DVS) may also be enhanced by use of a CM system which enforces appropriate access controls and audit mechanisms. The level of assurance required of all tools used is also a life-cycle issue: maintenance of satisfactory assurance may require keeping all tools under configuration management, for example, and the use of additional tools (such as subset-conformance checkers) to ensure that other tools (such as compilers) are only employed within the limits of their own assurance. Direct assurance of one tool, a compiler for example, may also be established through the use of another (a de-compiler or compiler validation suite). See Section 3.

4.1.3 Development (CC Assurance Class ADV)

The CC approach to development concentrates on establishing consistency between increasingly detailed levels of design representation. Assurance of this consistency can be facilitated by tools which maintain traceability between representations. Integrated development environments using semi-formal notations such as UML [13] can be used to support such a lifecycle, the rigorous separation of functional specification, high-level design, low-level design and implementation representation which (the current version of) the CC requires is not necessarily natural in such frameworks. See Section 4.4 for further discussion. The task of demonstrating correspondence between implementation and low-level design is facilitated by many of the software quality tools identified in Section 3: subset conformance, detection of run-time errors and software weaknesses all support this goal, as do some forms of object code verification.

4.1.4 Testing (CC Assurance Class ATE)

Some degree of test automation is likely to be used in any substantial product development, and any mechanisms which encourage the repeatable and controlled execution of tests will provide a degree of assurance in the design process. Some assurance benefits maybe expected from coverage analysis tools, although measurement of the proportion of a design which is exercised may not be a good prediction of the actual performance of the tests in detecting security-related errors. Management tools, such as configuration management and traceability tools, will also be applicable to tests and the test process.

4.2 For evaluators

The areas in which tools are directly applicable to assurance are perhaps more restricted than the general benefits of development tools, but the specific value which could potentially be obtained in some cases is nevertheless substantial. In the analysis, it proved useful to consider areas in which evaluators seek confidence, such as:

- Correct functionality is crucial, but in the majority of cases restricted to an informal review
- Identification of specific risky constructs, including error conditions, common vulnerabilities such as buffer overflows, and issues regarding concurrency.
- Consistency between design representations, and across interfaces between different elements.
- Sensitivities to platform and compiler attributes, which may become weaknesses if external dependencies change.
- General structure and behaviour of the program, as a prerequisite for assessing other issues, and also to illuminate information flow, for example.

Note that although these issues were examined from the perspective of an evaluator or assessor, developers are likely to use the same tools and techniques in order to reduce the risk that issues may be discovered later in the product life cycle.

4.2.1 Assurance of correct development (CC Assurance Class ADV)

The bulk of the information available to an assessor arises from the development process – any evaluation will therefore expect to make use of the tools discussed in the previous section. The applicability of other tools will depend largely on the nature of the information available: for medium and low-level assessment (the vast majority of cases) much of this information will be informal. In these cases, the key documentation (functional specifications, high- and low-level designs) may be natural language texts – there may be scope for the use of documentation tools such as readability metrics and indexing tools, but little true automation may be expected (NASA's Automated Requirement Measurement tool (ARM) is an interesting extension to an important class of documents [14].) Where semi-formal notations are used, mechanical consistency checks may be implemented (often as part of an IDE) ,but acceptance of such checks a assurance evidence is hampered by lack of commonly agreed representations and semantics for such checks.

In contrast, source code is by its nature formal and suited to the provision of mechanical support for the key assurance challenge of accessing and comprehending large quantities of technical information. Navigation and documentation aids (such as cross referencing and indexing tools) are important supports to assessment activities. Tools which provide some degree of abstraction (such as generating a call-tree or dependence graph) can be used to support comparison of the implementation with a low-level design, and can assist in identifying security-enforcing functions and their dependencies.

4.2.2 Assurance testing (CC Assurance Class ATE)

Assurance activities will typically include both an assessment of testing carried out during development and an element of independent testing. Both classes of activities will be facilitated by tools as discussed in the previous section (Section 4.1). The identification of specific vulnerabilities is also likely to involve testing in addition to the activities discussed in the next section.

4.2.3 Vulnerability identification (CC Assurance Class AVA)

The search for specific vulnerabilities is an essential element of security assurance. This is an area in which a number of specific vulnerability detection tools have been proposed (See Section 3) and their use is obviously a potentially valuable source of evidence, but the value of their results will be crucially dependent on parameters which are not necessarily easy to characterize, such as the proportion of identified problems which are not, in fact vulnerabilities (false positives) and the proportion of vulnerabilities which are present but not detected (false negatives). Similar concerns also apply to tools which look for general weaknesses which may be associated with breaches, such as run-time errors. The characterization and qualification of these tools is an important area of research (see also Section 3).

4.2.4 Manual assurance is essential

Although we have identified a number of areas in which tool support may support the assurance activities, there are a number of areas where no substitute to manual review and assessment is practical. This is the case, for example, in areas where the key attributes are the clarity and completeness of documentation, such as prevention of accidental misuse and installation and operational guidance generally. General purpose tools will also have limited use in some technical analysis, such as determination if the strength of security functions is appropriate.

4.3 Functional requirements

The discussion above has concentrated on assurance requirements – constraints on how a product is constructed – rather than the function it actually performs. In general, tools will not be able to confirm functional correctness of a product, although customised tool-supported analysis may be justifiable for some specific projects. There are some specific areas, however, where tool support can be valuable in assuring functional correctness, including:

- Control and data flow analyses. The more sophisticated program analysis tools can derive the flow of data and control through a program. This can be valuable in demonstrating the adequacy of various controls and policies, such as ensuring that security functions are invoked prior to any action which might compromise security (mediation).
- Failure mode analysis. Tools which detect vulnerabilities or general weaknesses in implementation provide information about the possible ways in which an implementation or function may fail. This is necessary to establish the appropriateness of measures which manage failures such as fault-tolerance or fail-secure functionality.
- Protocol and algorithm correctness. Although full proof of correctness of an implementation is not likely to be practical for the vast majority of security products, there are elements which, because of their extreme criticality or wide deployment, may be subject to more stringent constraints. In these cases, formal verification with specialist tool support may be appropriate. Typical applications might be security protocols or algorithms used by fundamental network infrastructure. ([15], [16], for example).

4.4 Evolution of the CC Evaluation Scheme

Our review identified a number of recommendations which we felt should be considered for changes to the CC and the evaluation methodology which advises how the criteria should be applied. The key recommendations regarding the methodology were:

- To require a search for known failure modes in the chosen implementation language, and mechanical enforcement of rules necessary to conform to well-defined language subsets.
- To link the sizes chosen for sampling activities to general software quality measures (allowing sample size reduction to be argued for developments showing demonstrably good quality).
- To encourage security targets to be defined according to practical application rather than to simplify evaluation.

Recommendations regarding the Common Criteria themselves were addressed more cautiously, because of the need to maintain consensus among all participants. The key suggestions were in the following areas:

- Fault elimination: Strengthening the functional specification of security functions to place greater emphasis on interfaces, and on the assumptions which the security functions make for correct behavior (e.g. integrity of memory, restrictions on flow of control). (Class ADV_FSP); inclusion in the development of evidence for the robustness of separation between security functions and other functions; and allowing tool support for maintenance of design representations, and relaxing requirements for strict refinement between design representations (while maintaining the necessary consistency).
- Fault discovery and removal: the requirement identified above, to search for known failure modes and insecurities, should ultimately be reflected in the CC.
- Failure tolerance. Designing systems to be tolerant of faults and failures is a crucial element in other product integrity domains, but is not emphasized in the CC. A requirement should be added to require analysis of possible failures and demonstrating that the design is appropriately robust.
- General changes: In other communities, standards-setting is moving towards a less prescriptive goal-oriented approach. The CC could be made less prescriptive, stating objectives of a successful evaluation and criteria which the recorded evidence must meet, but leaving open the means of meeting these objectives. This would facilitate competitive improvement of the evaluation process. To maintain mutual recognition in the light of this change, recognition should be based on establishing that different approaches are consistent in their effectiveness and findings, rather than that they produce identical results.

Since the completion of the work reported in [4], a new draft issue of the Common Criteria and the evaluation methodology has been published [17]. The new draft addresses many of the recommendations made here:

- Greater emphasis is placed on architectural integrity, and on demonstrating that other functions do not interfere with the security functions, although explicit failure mode analysis is not required.
- More explicit requirements are placed on specification of the interfaces of security functions.
- The development assurance family (ADV) has been revised to simplify the constraints placed on design documentation.
- The vulnerability analysis requirements include requirements to search for known classes of vulnerabilities.

The evaluation methodology remains too abstract to provide concrete advice on the use of tools, although clearly tool support will be advantageous for those activities which are required to be methodical.

Although the new version does significantly introduce noticeable simplification and significant re-structuring in many requirements, the majority of key assurance activities remain the same; the value and applicability of tool support will, therefore, remain unchanged.

5. CONCLUSIONS & FUTURE WORK

This paper takes results from number of existing studies:

- a goal-based view of the objectives of a secure product development [5],
- a review of the applicability of tools to security assurance [4] and
- Emerging work on taxonomies of software security tools [11],

and presents a summary which highlights where software tools may be expected to add value to development programs.

5.1 Applicability of Tools

Our review identified a number of areas where existing security evaluations could be supported by existing tools:

- Control of changes and configurations of products, product variants and assurance evidence
- Identification of general weaknesses, violations of coding standards and subsets, and potential run-time errors
- Identification of known vulnerabilities
- Assisting in an assessors understanding of a potentially large volume of potentially complex information

In the short term, use of tools in these areas appears most likely to improve the value of assurance (in terms of reduction of vulnerabilities discovered in service) rather than to decrease cost. We believe that it may be possible to achieve savings, primarily in the cost of developing the information required to support assurance, if increased use of automated document management and change control tools, and increased use of tools to enforce coding standards and subsets, were combined with a shift in the focus of development and evaluation processes.

Any successful deployment of tools will require that the tools themselves are adequately assured.

5.2 Varieties of assurance

The security of a product depends on many factors, and consequently can be improved and demonstrated by a range of different measures. The benefits of different assurance measures can be comprehended, and trade-off decisions facilitated, by considering their contribution to a structured assurance argument. Such an argument can provide a framework for planning assurance activities and identifying the support which tools can provide.

Some of the more important measures are specific to security assurance, such as searching implementations for known vulnerabilities, but we believe that systematic examination of the goals of a secure development demonstrates that more general assurance tools provide significant value in areas including general software quality, robustness of architectures, configuration management and change control. Our studies further indicate that for benefits in cost as well as quality, security

assurance must be an integral element of the development process, taken into consideration as key design decisions are being made.

5.3 Future Work

Although the opportunity for tool-supported security assurance is attractive, there are several questions which must be resolved if security assurance tools are to be widely adopted:

- Classification of tools and techniques, and development of common understandings of the value and function of each class, is necessary both to justify the adoption of tools, and to provide a basis for tool assurance. To support cost-benefit analysis, the classification must reflect benefits (eg risks reduced) rather than functional behaviour.
- Practical tools must be usable: issues such as consistent and informative output, reduction of false positive and false negative results, and scalability to large code bases are paramount.
- The qualitative discussion of assurance presented here must be refined to give quantitative cost-benefit arguments for tool adoption. A systematic approach to assurance will be required to allow tradeoffs to be made not only between the cost of assurance and the cost of failure, but between mechanisms which may each improve security in very different ways.

6. ACKNOWLEDGMENTS

The authors would like to acknowledge the support of CESG, Praxis High Integrity Systems Ltd, and all the stakeholders involved in the Common Criteria study [4]. Particular thanks are due to Keith Banks for his contributions to [4], and to Phil Core for his comments on drafts of this paper. The referees also provided useful comments.

7. REFERENCES

[1] U.S. Department of Homeland Security, The National Strategy to Secure Cyberspace, February 2003

[2] Anderson, R.J., Why Information Security is Hard – An Economic Perspective, *Proc. Annual Computer Security Applications Conference*, 2001

[3] Common Criteria for Information Technology Security Evaluation, Version 2.2, January 2004 Available from www.commoncriteriaportal.org

[4] Praxis Critical Systems Ltd, EAL4 Common Criteria Evaluations Study, September 2004, available from http://www.cesg.gov.uk/site/iacs/itsec/media/techniques_tools/eval4_study.pdf

[5] U.K Defence Procurement Agency and Praxis High Integrity Systems Limited, SafSec Project, www.safsec.com

[6] U.S. National Institute of Standards and Technology, SAMATE Project, http://samate.nist.gov

[7] Secure Software Inc., The CLASP Application Security Process, 2005.

[8] Kelly, T.P., Arguing Safety – A Systematic Approach to Safety Case Management, DPhil Thesis YCST99-05, Department of Computer Science, University of York, UK, 1998

[9] Common Methodology for Information Technology Security Evaluation, Version 2.2, January 2004

[10] Motor Industry Software Reliability Association, MISRA-C:2004 - Guidelines for the use of the C language in critical systems, ISBN 0 9524156 2 3, 2004

[11] U.S. National Institute of Standards and Technology, SAMATE Project Tools Survey, http://samate.nist.gov/index.php/Tools

[12] RTCA Inc, DO-178B, Software Considerations in Airborne Systems and Equipment Certification, 1992

[13] Booch, G., Rumbaugh, J. and Jacobson, I., The Unified Modeling Language User Guide Addison-Wesley 1998

[14] NASA, Automated Requirement Measurement (ARM), http://satc.gsfc.nasa.gov/tools/arm/

[15] Lowe, G., An Attack on the Needham-Schroeder Public-Key Authentication Protocol. *Information Processing Letters 56*, 3, 1995, 131-133.

[16] Reed, J.N., Jackson, D.M., Deinov, B., and Reed, G.M., Automated Formal Analysis of Networks: FDR models for arbitrary topologies and flow-control mechanisms, *Proc. Joint European Conferences on Theory and Practice of Software* LNCS 1382, Springer 1998.

[17] Common Criteria for Information Technology Security Evaluation, Draft Version 3.0, July 2005