TickIT International

The quarterly journal of the TickIT software quality certification scheme ISSN 1354-5884

Contents

IT 3Q06

IT security has been an issue exercising many nations over the past few years – we now have a definition of the Common Criteria that will allow the ready recognition of IT security and the level of assurance across the world. Fiona Pattinson and Mike Nash give us an understanding of how these criteria work and will follow this up with a detailed look at the assurance components of the criteria in our next edition.

Mike Forrester

Many a traditional software quality practitioner, like myself, put a great deal of reliance on the design review process and the success achieved in an 'ego-less' exchange amongst peers as to the adequacy of a piece of code design. When some of the Agile methods came on the scene we were alarmed at the apparent lack of this formal step! Well, 'pair programming' is a practice that can be applied to the likes of XP which results in better quality design and code – Jeff Langr explains how the natural resistance to this concept can be overcome to the benefit of software quality.

Our lives seem to be dominated in recent times by the juxtaposition of TickIT with CMMI and a discussion of whether one or the other should dominate. Well, the commercial reality is that neither should – what should dominate a company's systems are its own requirements. So a complex model emerges that has elements of TickIT, CMMI, ITIL, ISO20000, and many more needed to satisfy a company's requirements. The $64,000 question is: 'Can we construct and use that model to satisfy the competing factions?' – Andrew Griffiths believes he has the answer.

Talking of CMMI, I got to wondering what SPICE was up to nowadays – I noticed a recent press release about Automotive SPICE which I thought might be of interest to you. We will give more detail of how SPICE is doing in later issues.

You will recall Intellect's collaboration with the DTI and the NCC in promoting a code of best practice for SMEs a couple of years ago – we have an update on what is currently afoot at the NCC from Daniel Dresner and news of a new quality marque to boost confidence in the ICT supply chain.

All copy and letters to be sent to the editor, at the following address:

TickIT Office, BSI, 389 Chiswick High Road, London W4 4AL.
Tel +44 (0)20 8996 7427 / Fax +44 (0)20 8996 7429
email: tickit@bsi-global.com

Copy on disk or email please – Copy should NOT be sent to the publisher

Copy Deadlines:

December 23 for publication January 15
March 31 for publication April 15
June 30 for publication July 15
September 30 for publication October 15

For **advertising sales** contact
Tina Shorter, Firm Focus, Folia, Flowers Hill, Pangbourne, Berks RG8 7BD.
Tel +44 (0)118 984 3949 Fax +44 (0)118 984 2493
email: tina@firmfocus.co.uk

firm focus

Published by Firm Focus on behalf of BSI August 2006
www:firmfocus.co.uk

BSI British Standards

TickIT 3Q06

IT Security Assurance and Common Criteria

by Mike Nash and Fiona Pattinson

In this article we briefly introduce the Common Criteria (CC) standard and describe the operation of the evaluation and validation schemes for IT products based upon that standard. In the next issue we will describe the security assurance components of the standard in more detail. Our audience is intended to be those software quality and software process improvement experts who have an interest in the evaluation or certification of IT products using the CC or its ISO equivalent, ISO/IEC 15408. Our aim is to give a brief overview of the security assurance framework defined by the standard. For more detailed and complete information we suggest that the reader visit the Common Criteria portal at http://www.commoncriteriaportal.org/, or the web site for their national scheme. For the UK, this is located at http://www.cesg.gov.uk. In particular, we highlight that the security functional requirements are a vital part of the Common Criteria but are not discussed in any depth here!

What is Common Criteria?

Nearly all commercial off-the-shelf (COTS) information technology features security properties and provides security functionality for its operation within an organization's IT infrastructure, such as authentication and access control to enforce authorization requirements. The Common Criteria is an established, internationally accepted and locally mandated standards framework for assessing the trustworthiness of security functionality in information technology products. It offers a powerful tool for product consumers to specify security and assurance requirements for the technology used to implement information systems. One part of such an evaluation is to look at the relevant software and product assurance processes. The CC does not mandate any particular methodology or approach for assuring software quality, but it does assess those processes that are in use by a developer for the IT product under consideration.

A Brief Introduction to Common Criteria

The Common Criteria standard has evolved from the prior criteria for information security evaluation defined by various nations – such as the European Union's ITSEC, TCSEC (the famous Orange Book developed in the U.S.), Canada's CTCPEC and the U.S. Federal Criteria. It has been supported by all these nations due to the recognition that a common set of criteria offers real advantages to co-operating users of assured IT products.

The CC philosophy is to provide assurance based upon an evaluation (active investigation) of the IT product that is to be trusted. Evaluation has been the traditional means of providing assurance and is the basis for prior evaluation criteria documents. In aligning the existing approaches, the CC adopts the same philosophy. The CC process involves the assessment of the documentation and of the resulting IT product by expert evaluators, with increasing assurance based upon increasing emphasis on scope, depth, and rigour.

The CC does not exclude, nor does it comment upon, the relative merits of other means of gaining assurance. Research continues with respect to alternative ways of gaining assurance. As mature alternative approaches emerge from these research activities, they will be considered for inclusion in the CC, which is structured so as to allow their future introduction into the criteria.

The Common Criteria is a multi-part standard, and is intended primarily to be used as the basis for evaluation of security properties of IT products. By establishing such a common base, the authors hope that the results of an IT security evaluation are meaningful to a wider audience.

The Common Criteria was developed by the Common Criteria Development Board (CCDB), a committee originally set up by the major nations with existing national criteria. These nations have been joined by several other nations as signatories to the Common Criteria Recognition Arrangement (CCRA), an agreement which allows for mutual recognition of certificates produced under all schemes that are part of the arrangement (except high assurance evaluations). It requires the use of the Common Criteria and a Common Evaluation Methodology (CEM) as the basis for the arrangement. There are currently (July 2006) nine certificate-producing nations and a further thirteen nations that accept certificates issued under the CCRA agreement.

Even for nations and organizations that do not participate in the arrangement, the Common Criteria standards have been recognized as a key development in the security evaluation process. They have been generally accepted world-wide through international review and the publishing of the standards by the International Standards Organization such as ISO/IEC 15408 (parts 1-3) and ISO/IEC 18045.

Next column

The Common Criteria standards may also be used outside the arrangement agreed by the members of the Common Criteria Recognition Arrangement (CCRA), one example is the wider evaluation results mutual recognition scheme that is employed in the European region, originally developed by SOGIS[1]. A further example of a nation that has adopted the ISO/IEC version of the standards, but which has not joined the CCRA, is the People's Republic of China. Figure 1 shows the history and relationship of the Criteria within ISO and the CCDB.

The evaluation process establishes a level of confidence that the security functionality of these IT products and the assurance measures applied to them meet these requirements. The evaluation results may help consumers to determine whether these IT products fulfil their security needs.

The CC is also useful as a guide for the development, evaluation and/or procurement of IT products with security functionality.

The CC addresses protection of assets from unauthorized disclosure, modification, or loss of use. The

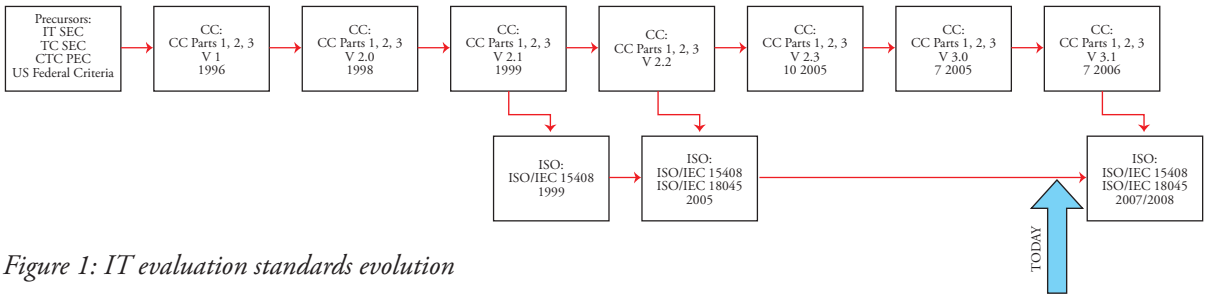


Figure 1: IT evaluation standards evolution

Under the CCRA (see Figure 2), nations that have a national scheme for conducting evaluations, run in accordance with the provisions of the CCRA, and approved under the terms of the CCRA, are called certificate producing nations. In July 2006, these were: Australia, Canada, France, Germany, Japan, Republic of Korea, The Netherlands, New Zealand, Norway, United Kingdom and United States of America.

Certificate consuming nations do not have a national scheme for conducting evaluations but have agreed to accept the certificates produced by the nations listed above. These nations are Austria, Czech Republic, Denmark, Finland, Greece, Hungary, India, Israel, Italy, Singapore, Spain, Sweden and Turkey.

The Common Criteria Paradigm

The CC permits comparability between the results of security evaluations conducted by different organizations in different countries. The CC does so by providing a common set of requirements for the security functionality of IT products (described in Part 2 of the standard) and for assurance measures applied to these IT products during a security evaluation (described in Part 3 of the standard). The functionality of these IT products may be implemented in hardware, firmware or software.

categories of protection relating to these three types of failure of security are commonly called confidentiality, integrity, and availability, respectively. The CC may also be applicable to aspects of IT security outside of these three. The CC is applicable to risks arising from human activities (malicious or otherwise) and to risks arising from non-human activities.

The CC is intentionally flexible, enabling a range of evaluation methods to be applied to a range of security properties of a range of IT products. Care should be exercised to ensure that this flexibility is not misused. For example, the CC should not be used to apply unsuitable evaluation methods, or to assess irrelevant security properties or inappropriate IT products, all of which could result in meaningless evaluation results.

CC Assurance

The CC describes assurance as "Grounds for confidence that an IT product meets its security objectives" (or at least the security objectives defined for the target of evaluation) and goes on to say "Assurance can be derived from reference to sources such as unsubstantiated assertions, prior relevant experience, or specific experience. However, the CC provides assurance through active investigation. Active investigation is an evaluation of the IT product in order to determine its security properties."

Evaluation has been the traditional means of gaining assurance, and is the basis of the CC approach. The CC philosophy asserts that greater assurance results from the application of greater evaluation effort, although the goal is to apply the minimum effort required to provide the necessary level of assurance. Evaluation effort depends upon:

1 The SOGIS agreement on the mutual recognition of certificates was originally based on the European ITSEC standard and adopted by Finland, France, Germany, Greece, Italy, The Netherlands, Norway, Portugal, Spain, Sweden, Switzerland and the United Kingdom. The arrangement was subsequently extended by these participants to include use of the CC at all levels of evaluation.

Next column

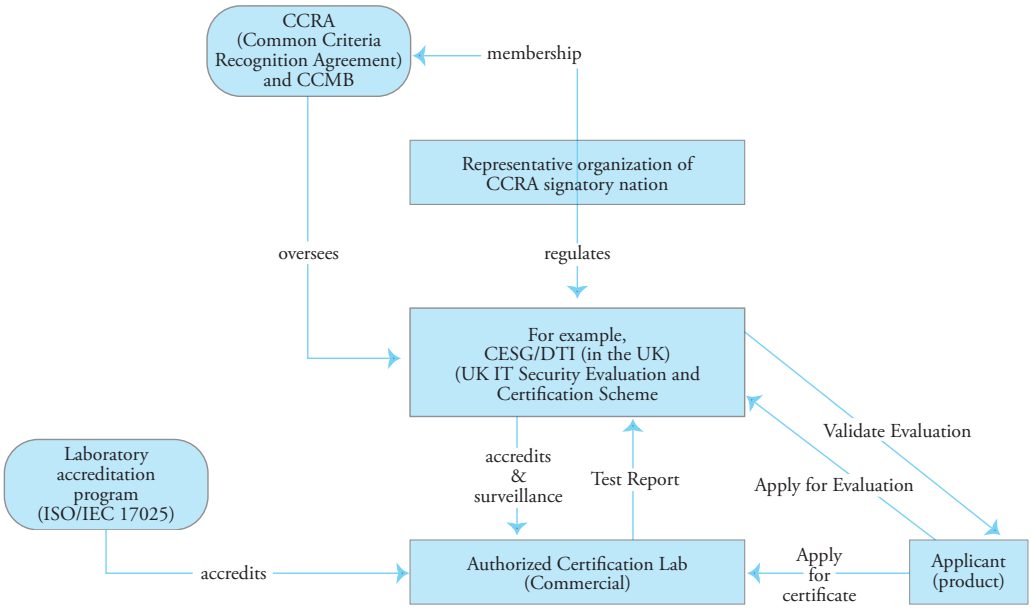


Figure 2: The evaluation and validation schemes of the CCRA

- scope –that is, the effort is greater because a larger portion of the IT product is included,
- depth –that is, the effort is greater because it is investigated to a finer level of design and implementation detail,
- rigour –that is, the effort is greater because it is applied in a more structured, formal manner.

Evaluation techniques that help create security assurance include:

- analysis and checking of process(es) and procedure(s),
- checking that process(es) and procedure(s) are being applied,
- analysis of the correspondence between design representations,
- analysis of the design representation against the requirements,
- verification of proofs,
- analysis of guidance documents,
- analysis of functional tests developed and the results provided,
- independent functional testing,
- analysis for vulnerabilities (including flaw hypothesis),
- penetration testing.

Security Assurance Requirements of The CC

For the remainder of this article, we will concentrate on Part 3 of the CC, Security Assurance Requirements (SAR). This part of the CC establishes a standard way to express assurance requirements for products; it specifies a standard layout and contents for the documents that define common requirements (called Protection Profiles within the CC), and those that define specific products to be evaluated (called Security Targets); it provides a mechanism for combining these assurance requirements into packages, and it defines some standard assurance packages called evaluation assurance levels (EALs).

The CC describes the Security Assurance Requirements (SAR) using a class and family structure. An overview of the structure is given in Figure 3 – Assurance class/family/component/element hierarchy.

Common Criteria Assurance Requirements

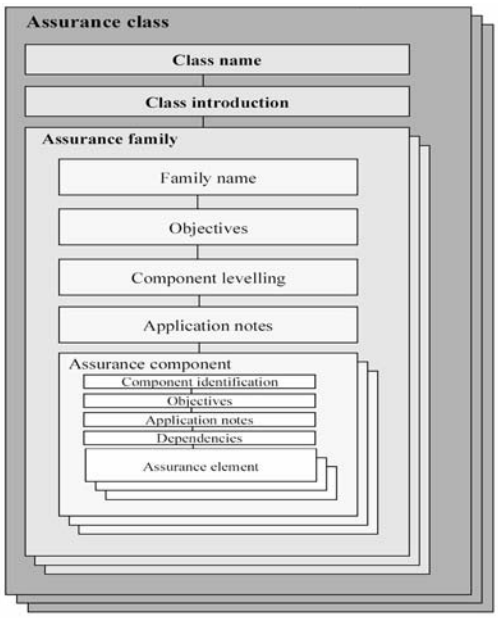


Figure 3 – Assurance class/family/component/element hierarchy

Each class addresses a particular aspect of assurance. Within each class, the criteria are broken down into families of related evaluation criteria, called components. The components within a particular class have a clearly defined relationship. For example, in Figure 4, the class as shown contains a single family. The family contains three components that are linearly hierarchical (that is, component 2 requires more than component 1, in terms of specific actions, specific evidence, or rigour of the actions or evidence). At present, the assurance families in CC Part 3 are all linearly hierarchical, although the CC notes that linearity is not a mandatory criterion for assurance families that may be added in the future.

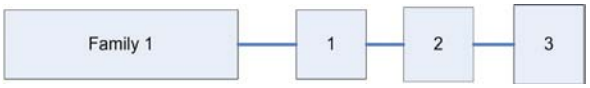


Figure 4 – Sample class decomposition diagram

Evaluation Assurance Levels (EAL)

An evaluation assurance level is a predefined package of evaluation components, describing a standardized evaluation requirement in terms of scope, depth, and rigour. A very generalised and informal description of the EALs, and reflecting our field experience in their use, is shown in Table 1.

In practice, all evaluations are based upon one of these EALs, sometimes augmented with a few additional components to address specific assurance requirements. In the next part of this article we will describe the assurance components defined in CC Part 3 in greater detail.

Acronyms

CC	Common Criteria
CCDB	Common Criteria Development Board
CCMB	Common Criteria Management Board
CCRA	Common Criteria Recognition Agreement
EAL	Evaluation Assurance Level
PP	Protection Profile
SAR	Security Assurance Requirement
SFR	Security Functional Requirement
SOGIS	Senior Officials Group for Information Security of the European Commission
ST	Security Target
TOE	Target of Evaluation
TSF	Toe Security Function

EAL1: functionally tested – The lowest level defined in the CC, achievable without access to developer documentation. In practice, EAL1 evaluations are hardly ever carried out.
EAL2: structurally tested – Represents the best that can generally be achieved without additional work by the developer.
EAL3: methodically tested and checked – Allows a conscientious developer to benefit from positive security engineering design without alteration of existing reasonably sound development practices.
EAL4: methodically designed, tested, and reviewed – The best that can be achieved without significant alteration of current good development practices. This EAL is typically cited as the highest level generally achieved by commercial software.
EAL5: semiformally designed and tested – The best achievable via pre-planned, good quality, careful security-aware development without unduly expensive practices.
EAL6: semiformally verified design and tested – A 'high tech' level for (mainly military) use in environments with significant threats and moderately valued assets.
EAL7: formally verified design and tested – The greatest amount of evaluation assurance attainable whilst remaining in the real world for real products. EAL7 requires formal modelling and is very expensive and resource intensive to complete successfully. In practice very few EAL7 evaluations have been performed, of products with severely limited functionality.

Table 1: Informal description of Evaluation Assurance Levels

References

- CCMB. Common Criteria Portal. Available from http://www.commoncriteriaportal.org/.
- The Common Criteria Sponsoring Organizations. "Common Criteria for Information Technology Security Evaluation Version 2.3: Part 1: Introduction and General Model." , 2005.
- The Common Criteria Sponsoring Organizations. "Common Criteria for Information Technology Security Evaluation Version 2.3: Part 2: Security Functional Requirements.", 2005.
- The Common Criteria Sponsoring Organizations. "Common Criteria for Information Technology Security Evaluation Version 2.3: Part 3: Security Assurance Requirements.", 2005.
- The Common Criteria Sponsoring Organizations. "Common Methodology for Information Technology Security Evaluation : Version 2.3 : Evaluation Methodology." 2005.
- (ISO JTC 1/SC 27), International Organization for Standardization. "ISO/IEC 15408-1:2005 Information Technology – Security Techniques – Evaluation Criteria for It Security – Part 1: Introduction and General Model.", 2005.
- (ISO JTC 1/SC 27), International Organization for Standardization. " ISO/IEC 15408-2:2005 Information Technology – Security Techniques – Evaluation Criteria for It Security – Part 2: Security Functional Requirements." , 2005.
- (ISO JTC 1/SC 27), International Organizationfor Standardization. " ISO/IEC 15408-3:2005 Information Technology – Security Techniques – Evaluation Criteria for It Security – Part 3: Security Assurance Requirements." , 2005.
- (ISO JTC 1/SC 27), International Organizationfor Standardization. "ISO/IEC PDTR 15446: Guide on the Production of Protection Profiles and Security Targets", 2000

Mike Nash has a long background in security evaluation criteria. His first involvement came in 1985, working initially within NATO using the US TCSEC 'Orange Book', and then setting up and managing the first UK evaluation facility. He helped develop the UK national criteria, the ITSEC and finally the Common Criteria. On the other side, he also advised major vendors and customers how to prepare for and successfully achieve evaluation. He is currently the Secretary of ISO/IEC Subcommittee 27 Working Group 3, which is responsible for standardization of security evaluation criteria. He is also the ISO Project Editor for Part 2 of ISO/IEC 15408, dealing with Security Functional Requirements. He is a consultant to several evaluation facilities and evaluation schemes.
Dr. Nash is a Director of Gamma Secure Systems Limited.

Fiona Pattinson is a laboratory manager for the information security provider, atsec (www.atsec.com). atsec has acredited laboratories for evaluation of requirements of Common Criteria under both the German (BSI) scheme and the US (NIAP) scheme, and for FIPS 140-2, and Personal Identity Verification. atsec also provides services for ISO/IEC 27001 and in IT security consulting.
Ms Pattinson is a Certified Information Systems Security Professional (CISSP) and Certified Software Development Professional (CSDP). She earned her M.Sc. in 'Computing for Commerce and Industry' from the U.K.'s Open University. She serves on the U.S. INCITS CS1 committee 'Cyber Security' and works on the ISO SC27 WG3 'Security Techniques' working on Common Criteria and other security standards.
Ms Pattinson's IT career began in 1984 with assembly programming, progressing through operations management, technical support, systems analysis, new-business development and as a software-development quality and information-security consultant.

Pair Programming

By Jeff Langr

Pair programming is one of the most contentious practices of extreme programming (XP). The basic concept of pair programming, or 'pairing', is two developers actively working together to build code. In XP, the rule is that you must produce all production code by virtue of pairing. The chief benefit touted by pairing proponents is improved code quality. Two heads are better than one. Note that pairing is a practice that you can use exclusively of XP.

However, a large number of developers despise the notion of having to sit next to someone for the better part of their work day. Reasons for resistance can include:

- the belief that pairing is a waste of time and ineffective,
- the belief that pairing should only be used occasionally,
- fear of exposing personal weaknesses,
- personality issues, including introversion.

Pairing is not for everybody. But too many people resist pairing based on a knee-jerk reaction to what they understand it to be. Usually, the practice and its benefits are not fully understood.

Say "pair programming" to a programmer and he'll probably frown or turn his back on you. But add some rules the programmers must follow--rules that help maintain each person's sanity--he just might come to find this practice rewarding and beneficial. This article, reprinted from Jeff Langr's Web site, explains the rules and how certain teams have reacted to this structured version of pair programming.

Pairing Technique

There are only a few simple rules to follow if you choose to do pair programming.

First and foremost: all production code must be developed by a pair. Conversely, this means that are plenty of other activities that you can undertake in the absence of a pair:

- work on any non-production code, for example: the acceptance test framework, other tools, build scripts, and so on; in most shops, these are in dire need of attention,
- work on documentation,
- work on spikes for future stories,
- learn how to use a new third-party product; learn a new coding technique; and so on …
- identify problem spots in the production code that need refactoring,
- refactor tests,
- improve the existing test coverage if necessary.

Any of the above could be done better if a pair were available, but they are usually lower risk activities. If time is absolutely a factor, the non-pairing developer can work on production code, but with the insistence that such code is peer-reviewed after the fact.

Having a non-pairing developer work on production code should be the exception, not the rule. As such, it should be justified and treated as a high risk activity.

The second pairing rule: it's not one person doing all the work and another watching. During a good pairing session, the keyboard should be moving back and forth between the two participants several times an hour. The person without the keyboard should be thinking about the bigger picture and should be providing strategic direction. They should also be helping to ensure maximal code quality and minimal defects.

Third: don't pair more than 75% of your work day. A good, productive run of six hours of software development is mentally exhausting. Make sure you take breaks! Get up and walk around for a few minutes at least once an hour.

Finally: you need to switch pairs frequently. Working with any one person for any extended duration will not only drive you nuts, but you will begin to lose the benefit of getting a fresh outlook on problems. Minimally, switch pairs at least once a day. In fact, I promote switching pairs once in the morning and once in the afternoon. In addition to getting new insight on solving a problem, there is another, less obvious, benefit to frequent pair switching.

Context Switching

When you sit down to work with a new pair, you must switch mental contexts from the task you were working on to a whole new problem. Context-switching is difficult. Work for only 55 minutes then switch tasks? Seems outrageous. One might think, "It'll take almost that amount of time for me to come up to speed on what you've just developed!".

An XP rule of thumb is: when something is difficult or painful, do it more often until it becomes easier. If integrating is a royal pain, do it more often until you learn how to do it better, or at least until it's apparent you've hit the point of diminishing returns.

If context-switching takes too much time, do it

Next column

more often. In theory, what this should do is force developers to write better code. By 'better' I mean code that accommodates cheap maintenance without adverse impacts on the system.

If it takes me thirty – or even ten – minutes to come up to speed on a task, yes, there is a thrashing problem. If instead the other developer has followed good design/coding guidelines (small, composed methods, no duplication, appropriate naming, and basic OO design principles go a long way), context switching should take only a couple minutes. Combine that with a test-driven approach, and I can quickly focus on a small amount of detail.

The Numbers

So how many people actively resist pairing? At a large shop of ~300 developers (divided among several teams), about fifteen (5%) of developers actively resisted pairing when they embarked upon XP. The rest of the people fell into one of three groups: skeptics, interested adopters, and sheep, divided fairly evenly in terms of numbers. After pairing for a few iterations, perhaps five of the fifteen resisters learned to enjoy it. Another five didn't mind it enough to complain any more, and another five hated it even more than before.

Having consulted in a good number of XP shops, my personal experience shows these percentages to be pretty consistent. Based on what I've seen, you'll end up with from one to five percent of developers who can't or won't pair. For most shops, that's one or two people.

Obviously there are always people who don't voice their objection and just go along with whatever is tossed their way. You do want to ensure everyone has a forum for feedback. I've solicited feedback via anonymous 3x5 cards, email, public forums, one-on-one discussions, however I could. Beyond that, if someone isn't going to be honest enough to complain, I suspect it says something about the caliber of that employee.

What To Do With These People

Any shop embarking on XP, RUP, Scrum, or whatever, needs a coach to steer people in the right directions. Engaging in a new process without a coach is worse than trying to play a football game without a coach – at least the football players have done it all before and know some of the things to watch for.

It's a coaching failure if I'm unable to turn some of the pairing resisters around. And usually I can, by working directly with them, by demonstrating the benefits firsthand, and by ensuring that the process otherwise goes smoothly. From my own exposure to pairing, I initially thought it was a bad idea. After practicing it a bit, I didn't necessarily love it, but recognized the benefits and was willing to do it. Shortly thereafter, I began to love what I was able to get out of it.

Most sizeable shops have small adjunct efforts. For valuable resources, one-offs are a great place. There's also the possibility of working on non-production code, such as tools for internal use.

Resisters might still be able to work within a team, as long as they put up with their end of the bargain. Remember that pairing is initially a way of doing continuous review. In lieu of pairing, the resister must initiate Fagan inspections or some other sort of formal review. One way or another, the code must be reviewed – otherwise you'll get pricy consultants (like myself) or unguided novices, producing unmaintainable garbage. Often, people find that the evil of pairing is preferable to the evil of group review.

Ultimately the rules should be up to the team, as long as the rules satisfy the requirement that code is reviewed. If the entire team revolts and insists upon no pairing, then they can all do inspections or whatever review form acts as a second-best choice. If 95% of the team insists upon pairing, and the sole remaining developer can't deal with it at all, the organization should help them find something else to work on.

While this may sound intolerant, remember that software development is a team effort. Suppose I go to your shop and find out that you value RAD. You sit in a board room with 25 other people for two days and hash out every detail of the project. You then produce 200 pages of design documents. But I have a personality disorder that prevents me from contributing in such an environment. I can't stand sitting in a room for days on end slogging through stuff, 95% of which is useless to my role. And I also have a disorder that prevents me from understanding design document doublespeak.

Do I belong in this organization? Probably not. If the organization is successful with this culture, why should they waste time and money trying to accommodate my abstinence?

I'm not trying to be clever or obtuse here. The point is that organizations should grow the cultures that they value, and that those cultures may not be appropriate for everyone. Anyone who says every shop should do XP or RUP or whatever is insane. The reality is that there are always other shops to choose from.

Other Pairing Benefits

The book *Pair Programming Illuminated* (by Laurie Williams and Robert Kessler, Addison Wesley Longman Publishing Co) goes into much further depth on the costs and benefits of pair programming, in addition to many other related topics. I highly recommend it. Over the years, I've built my own brief list of pairing benefits.

Next column

General benefits:

- Produces better code coverage. By switching pairs, developers understand more of the system. Many benefits can result from this increased knowledge.
- Minimizes dependencies upon personnel. Everyone worries less about buses and trucks.
- Results in a more evenly paced, sustainable development rhythm.
- Can produce solutions more rapidly.
- Moves all team members to a higher level of skills and system understanding.
- Helps build a true team.

Specific benefits from a management standpoint:

- Reduces risk.
- Shorter learning curve for new hires.
- Can be used as interviewing criteria ('can we work with this guy?').
- Problems are far less hidden.
- Helps ensure adherence to standards.
- Cross-pollination/resource fluidity. Allows you to swap members from two or more teams on occasion, with minimal downtime. Usually each person will bring immediate value to the new team ('you guys should use this utility class that we built here...').

Specific benefits from an employee perspective:

- Awareness of other parts of the system.
- Resumé building.
- Decreases time spent in review meetings.
- Continuous education. As someone who thinks he is a pretty hot programmer, I still learn new things every day from even the most junior programmers.
- Provides the ability to move between teams. ('this team is boring,' 'I can't stand working with him,' 'that stuff they're doing looks cool'). Since this can be a benefit for management as well, they don't have to clamp down on their resources.
- More rapid learning as a new hire. You don't sit and read out-of-date manuals for a week, or worry that you're going to be fired because the system looks indecipherable.

These benefits come about from monitoring the process, making sure the technique is executed well, and fixing problems. Don't forget a coach!

Skill Level Gaps

An experienced developer can outperform an inexperienced developer by two, three, five, ten or even twenty times. Sitting with a novice can be excruciatingly painful. But I'd rather burn a little time bringing other guys up to speed as soon as possible. It pays off in spades.

Pairing allows me to keep tabs on their work and make sure they are not producing junk that will have to be rewritten. It also prevents them from holding up the project. Several years ago I was on a project that ultimately got cancelled, largely due to a *schmuck* that couldn't get his work done in time, and when it did get done, it was garbage. With a pair, incompetence that can destroy a project surfaces far more quickly.

I still learn some very interesting things from working with novice developers. And they learn a wealth of things that they would never learn were they to be left to their own devices.

Leaving an inexperienced developer alone to suffer through the system and other issues presents them with a steep learning curve. Through pairing, this learning curve begins to flatten more early on in the project. The more time I spend up front with a novice developer, the more we can depend upon their contributions later in the project – when it matters far more.

Of course, the novice must be capable of growing. Pairing lets you find out quickly who's worth keeping versus who will always be a drag on the team. Management can get involved in the first month or so of a project, as opposed to late in the project when it's crisis time.

Final Comments

I've been in the position of promoting XP and hence pairing as a consultant for a while now. Until I did a longer, six-month consulting stint, my pairing experience was more sporadic. Sure, I saw the benefits and the negatives, and did it enough to know how to make it work. Plus I learned plenty more about it from other sources. But until I sat there and actively paired for a longer duration, some of its nuances weren't as evident.

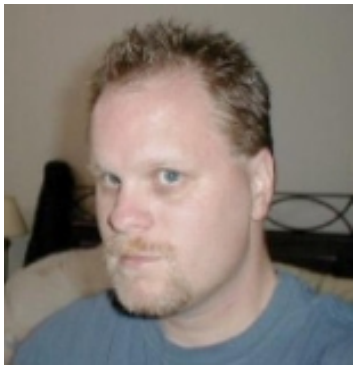
In fact, pairing seemed to me like a nice thing to promote, but maybe pairing wasn't something that I needed to worry so much about. Not drinking my own kool-aid!

What I realized is that pairing after a while becomes a dependency, in both a good way and a bad way. I learned to look forward to most pairing sessions (there are always some difficult people). I also felt naked when not pairing, and began to question more what I produced by my lonesome self. Pairing became assuring and thus relaxing.

Before, I would be overly confident that I was a great programmer and that I produced code that was just fine. Maybe not! A few pairings with some sharp people and I learned a few cool new techniques. I took on more humility.

Dependencies in code can be bad, as can dependencies in life. The secret is in managing these dependencies well. Learn to use pairing as a tool to help you do better the next time you aren't.

Next column



Jeff Langr has been loving and living software development since 1982. He's the author of two books, *Agile Java: Crafting Code With Test-Driven Development* (Prentice Hall, 2005) and *Essential Java Style* (Prentice Hall, 2000), as well as over twenty published articles. He's currently working with Bob Koss to write a book that shows how to do test-driven development in C++. Jeff helps development teams improve their productivity by mentoring and training through his company, Langr Software Solutions (http://langrsoft.com).

The Marque of the West Midlands

by Daniel Dresner

There's Corgi for the gas installers and the Federation of Master Builders. Sometimes these marques can be a helpful starting point when personal recommendations are few and far between. But what about the suppliers of ICT services? What marque of reliability builds the trust in the small local firm that can cable your building, or give you a tailored point-of-sale system that will allow your small shop to expand?

Advantage West Midlands (a regional development agency) recognized that there is an opportunity to improve local services for customers of ICT and boost business opportunities for those SMEs who supply it. AWM briefed the National Computing Centre to build a consortium of purchaser and supplier stakeholders to assess the opportunities for a new quality marque to boost confidence in the ICT supply chain. The objective is to create an environment of trust that helps ensure a fair and rewarding price for the suppliers to charge, and value for money for the customers.

From this project an ICT supplier standard is emerging to which ICT suppliers can be accredited. It will be available to any size of ICT supplier, but will have a particular emphasis on helping SMEs, who represent 98% of the market.

The standard is being developed to be wholly complementary to established work including (to name but a few) ISO 27001 (security), ISO 9001 (quality), ISO 15288 (Systems lifecycles), ISO 15504 (process assessment) and, of course, our own dear TickIT. In fact, it's not just complementary; it positively encourages a greater interest in these standards.

Businesses will be assessed on three pillars of the right processes being realized by the right people, and resulting in the right level of performance. The standard – already in its first draft – looks at essential processes, the retention of staff to deliver the business benefits of the opportunities that arise, and a scorecard of performance measures that shows the effectiveness of all that activity for the supplier and its customers.

The standard will be published as a general benchmark for all ICT suppliers with appendices for specific segments of the industry. This recognizes the differing skills and competencies of the reseller and the product/service developer.

This work is extremely timely and dovetails with NCC's alliance with the BCS, Intellect, and e-SkillsUK on the professionalism in IT agenda.

http://www.isprofessionalism.org.uk/

It is a pragmatic piece of work which is thoroughly designed and reviewed by the stakeholders.

http://www.ictss.org.uk/

TickIT International will continue to chart its progress.

Considering How to Reduce the Strain of Multiple Models

By Andrew Griffiths

How the world has changed over the last 24 months. The cautious return to investment in software for strategic advantage is gathering pace and with it the associated investment in process improvement. The level of investment is not the only thing to have changed; the focus on process improvement models is sharper than ever, with CMMI and ITIL being the two strong growth areas. Where does this leave ISO 9000 and TickIT? Interestingly enough, still very much in the game but with a dent in market share.

Before we go any further what's CMMI? (This is the very short version.) To quote the SEI's website the CMMI: "Capability Maturity Model® Integration (CMMI) is a process improvement approach that provides organizations with the essential elements of effective processes". In some respects the CMMI can be described as a set of tools: a model of best practice against which to review your processes; a model of how to improve your processes; and an associated appraisal method to assess an organization. The SEI provides a framework for ensuring the quality of the delivered appraisals and training (through SEI Partners; beware there are non-SEI approaches for CMMI that are unregulated).

The model consists of Process Areas (PA). Each process area comprises a set of practices that direct improvement. There are two main 'representations' of the model which embody different approaches to implementing process improvement. The 'staged' representation provides a structured route through the improvement process. The 'continuous' representation enables selection of an organizational-specific route for the process improvement.

The SEI-sanctioned appraisal method is the SCAMPI (Standard CMMI Appraisal Method for Process Improvement). Three different classes of appraisal are available. A Class 'A' appraisal is the formal means of determining the level to which the model is satisfied. It delivers a highly rigorous, and reliable means of assessing the organization. Results of this class of appraisal can be posted on the SEI's website. Lighter approaches are available through Class 'B' and 'C' appraisals. The range of classes of appraisal addresses the different needs of an organization dependant upon where it is in its improvement journey: the formal check, the robust baseline and the quick look.

Next column

Why the Interest in Multiple Models?

This is a complex question that has several dimensions depending on the nature of your business but some of the key drivers are:

- The market penetration of offshore services from India where various process improvement (PI) models have been used to support the sales and marketing process, most notably CMM/CMMI. This has led to an increased awareness of these models in UK plc, thereby driving adoption locally.
- Many global companies and government bodies run open competitive tendering processes; these can often result in a larger volume of responses that can be time-consuming and expensive to evaluate. The insertion of a few quality standards to act as a qualification bar (even if there is no substantive internal interest in the specific standards) is becoming commonplace.
- The use of supplier assessments for strategic procurement by, or on behalf of, purchasers, is becoming more common with CMMI (outside of US DOD), BS ISO/IEC15288:2002 and BS ISO/IEC15504:2003 gaining significant ground.
- The response of system integrators and software companies to client interest in PI models has been to look to achieve against these models to avoid disqualification from future procurement.
- There seems to be a clearer understanding in the market that the various PI models are useful, but for specific purposes and scope.

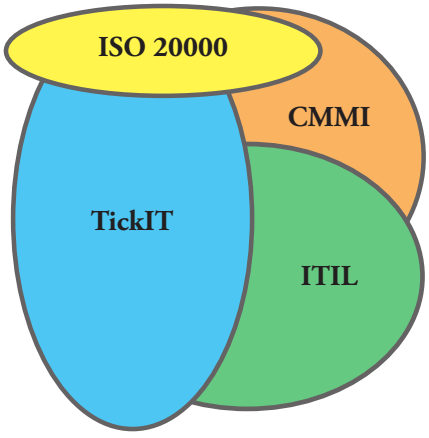


Figure 1: Multiple Models

Often one model in isolation will not meet an organization's improvement objectives and the set of models chosen is usually driven from three main perspectives: systems and software engineering, operability and governance.

So What's the Problem With This?

Let's imagine a real organization where they are using CMMI + COSO + TickIT to address their business needs. (Note: COSO is The Committee of Sponsoring Organizations of the Treadway Commission, a US financial governance organization.) This would generate multiple appraisals on the organization in question, with some groups being reviewed on every appraisal. (Just before anyone mentions it, there is an option in the SCAMPI method to run an ISO15504-compatible appraisal). What if the organization is a System Integrator with a diverse customer base and markets? It is likely that a few more PI models would be added into the mix to cater for the demands of specific clients. This is building a very real, overlapping appraisal overhead that, when you roll up all the involved people, is beginning to look too large.

What can be done?

Well, as ever, these useful models come from different organizations, which provide little or no integration and certainly no cross-model appraisal approaches. Clearly a single appraisal could be completed to address some of these models together (for example, CMMI, ITIL, TickIT) but the governing organizations would almost certainly not recognize any results of such an appraisal. However, I think it may be simpler than it appears. The SEI's SCAMPI appraisal method is executed to a high standard of consistency and requires more supporting evidence in the process than the other models (many do not even have a matching appraisal method). The SCAMPI method requires a model organized in a particular way (process areas, generic goals, specific goals and so on) that is not the way the other models are organized. So, work is required to integrate the models required for the appraisal together. To provide a specific example, let's take CMMI + ITIL – in true 'Blue Peter' fashion I have one I prepared earlier (actually one Kieran Doyle prepared earlier, to be accurate).

The integrated model of CMMI + ITIL (Figure 2) was created by examining the overlap between these two models and creating additional CMMI process areas to cover the components of ITIL uncovered, the operability management category. Also, a number of small additions were made to specific practices in the CMMI where extending the process area was more appropriate. Some of the differences in emphasis between CMMI and ITIL create some friction, such as completing causal-based analysis is a much earlier requirement in ITIL than CMMI; the easiest way to reconcile this is to use the continuous representation. Staged is not impossible, but much more work would be required. Before I continue I must emphasise that this approach is a proposed approach, not once sanctioned by the SEI!

CATEGORY	PROCESS AREAS							
PROJECT MANAGEMENT	Project Planning	Project Monitoring & Control	Supplier Agreement Management	Risk Management	Integrated Teaming	Integrated Project Management	Quantitative Project Management	Integrated Supplier Management
ENGINEERING	Requirements Management	Requirements Development	Technical Solution	Validation	Verification	Product Integration		
SUPPORT	Configuration Management	Measurement & Analysis	Process & Product Quality Assurance	Decision Analysis & Resolution	Causal Analysis & Resolution	Organizational Environment for Integration		
PROCESS MANAGEMENT	Organizational Process Focus	Organization Process Definiton	Organizational Training	Organizational Innovation & Deployment	Organizational Process Performance			
OPERABILITY MANAGEMENT	Strategic Relationship Management	Budgeting & Accounting	Service Level Management	Service Continuity & Availability	Incident Handling			

Figure 2: CMMI + ITIL – an integrated view

We now have a strong appraisal method, with a model that we can demonstrate covers a combined CMMI + ITIL scope. So all we have to do is run a SCAMPI A appraisal against this and we have a CMMI + ITIL result, right? Well, no. The 'small additions' to the base CMMI model would certainly jeopardise acceptance of the results from the SEI perspective. Covering the additional process areas adds complexity too, but this is a secondary issue. Given SCAMPI is not the way to conduct an ITIL appraisal, and the appraisal has been conducted against a combined model, I can foresee similar problems here.

So why tell you all this? Simply, this approach probably cuts an acceptable compromise that recognizes some of the 'political' issues (from a model perspective). Additionally I expect to test this theory more rigorously over the next twelve months. I hope my pragmatic optimism is well placed.

Andrew Griffiths joined Lamri in 2003 as Managing Director: bringing his experience of large-scale process improvement, RUP deployment and out-sourcing to the Lamri team. Andrew has worked in process improvement since 1994 using many tools and techniques including CMM, CMMI, Booch, UML, DSDM, RUP, Objectory and various development tools across the lifecycle. He is a regular speaker at conferences and seminars on topics including the application of CMMI, Off-Shoring, Programme and Architectural Governance.
www.lamri.com
andrew.griffiths@lamri.com

Automotive SPICE Gets Motoring

I got to wondering what SPICE was up to nowadays and in the recent press I noticed a press release about Automotive SPICE which I thought might be of interest to you: Wipro Technologies, the Global IT Services Division of Wipro Limited (NYSE:WIT) has announced that its Automotive Group has achieved Automotive SPICE …'

Wipro Technologies, the Global IT Services Division of Wipro Limited (NYSE:WIT) has announced that its Automotive Group has achieved Automotive SPICE™ Organizational Maturity Level 5 certification under the PATHFINDER™ interim assessment and certification scheme for process capability and organizational maturity.

Wipro Technologies Automotive Group is the first organization in the world to achieve Maturity Level 5 and is the first organization to achieve certification under the PATHFINDER™ scheme claims the company.

The Automotive SPICE™ initiative includes car manufacturers such as AUDI AG, BMW Group, DaimlerChrysler AG, Fiat Auto S.p.A., Ford Werke GmbH, Jaguar, Land Rover, Dr. Ing. h.c. F. Porsche AG, Volkswagen AG and Volvo Car Corporation.

A joint assessment team from Impronova AB (Sweden) and KPMG (India) performed the PATHFINDER™ assessment. The assessment team leader was Alec Dorling from Impronova AB. Impronova AB is an approved Ford Software Quality Partner.

"We went through rigorous review and analysis of Wipro's automotive software methodology and we are pleased to award the AutomotiveSPICE Level 5 certification to Wipro. We are sure that this certification coupled with Wipro's stringent quality standards will be of great value to car manufacturers and the tier 1 automotive suppliers in meeting the challenge of developing embedded software" said, Alec Dorling, Automotive SPICE assessor, Impronova, Sweden.

The scope of certification includes the design, development, conversion/porting, maintenance and testing of software systems in the automotive domain.

Automotive SPICE™ has developed a common framework for the assessment of suppliers in the Automotive Industry through the publication of the Automotive SPICE™ Process Assessment Model and Process Reference Model.
For more information see: www.wipro.com/