

TickIT International

3

It is now getting very close to the initial launch of **TickIT^{plus}**, so it is important for all TickIT-registered organizations to be fully aware of what is happening. We are therefore glad that Derek Irving has provided another update on the progress being made.

4

Software Asset Management is becoming of ever greater importance; you will recall we have been tracking developments with articles from Dave Phillips in 2Q08 and in 1Q09. Well, things have moved ahead and are taking shape, so in this issue we have a major article from Steve Klos, who is the convener of ISO/IEC 19770-2 which is nearing completion ready for publication. Steve gives us a thorough run-through of the issues surrounding the generation of universal Software ID Tags and their future role in Software Asset Management.

18

In our business of quality assurance I find it useful, every now and again, to go back to basics and take another look at what we do for a profession. I therefore thought you might like to see a paper from Dr Jeffrey Voas, who 'unpeels' the various layers that make up that thing called software quality, resulting in that most important quality of 'fit for purpose'.



Mike Forrester



IT 4Q09

TickIT International

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

This publication is a means of communication with all TickIT-registered organizations and TickIT-minded individuals throughout the world. It also acts as an information exchange and sounding-board for anyone committed to IT quality.

The editorial team always welcomes input and comment.

As editor I will invite comment and debate on specific issues within each edition so as to develop thinking and promote excellence within our industry.

To make comment or provide input please to e-mail the team.

[click here](#)

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 7424 / 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, 26 Meadowside Road, Pangbourne, Berks RG8 7NH.

Tel +44 (0)118 984 3949 Fax +44 (0)118 984 2493

email: tina@firmfocus.co.uk



Published by Firm Focus on behalf of BSI October 2009
www.firmfocus.co.uk



© BSI and contributors 2009. Opinions expressed in this publication are not necessarily those of BSI.



TickIT^{plus} – the future of TickIT: Project Progress, October 2009

This is the quarterly status report on the development of the **TickIT^{plus}** scheme, which will completely change the way that IT is covered under ISO 9001-accredited certification. As previously described, both **TickIT^{plus}** and the existing TickIT scheme are run by the Joint TickIT Industry Steering Committee (JTISC) with administration from BSI. JTISC is formed of several major IT suppliers, users and industry bodies. **TickIT^{plus}** is a part UK Government funded extension of the original scheme, aimed at improvement and wider use, and which will add capability assessment and an extended standards scope as well as broadening auditing skills and training. More details and information on how to contribute comments can be found at our new website – see below for details.

Overall Progress and Development

The technical details of the scheme are now almost complete, with the final specification drafts about to be approved. The first two pilot training courses, which will enable existing TickIT Auditors working for some of the principal Certification Bodies to be registered Foundation grade **TickIT^{plus}** Assessors, will be delivered in October. This will be accompanied by the Foundation examination. We are now in the final stages of negotiation with the Global Association for Software Quality (GASQ) for the registration of **TickIT^{plus}** Assessors and Practitioners, accreditation of training providers and delivery of the independent examination services.

The first of the scheme's documentation – '*Delivering Quality in IT*' – is about to be published by BSI, and the **TickIT^{plus}** website, which until now has had a temporary hosting arrangement, is about to be taken over and managed, also by BSI.

Scheme Launch

For a number of reasons – the time needed to bed the scheme in, integration with the migration plans from the existing TickIT scheme and various financial and organizational constraints – it has been decided to extend the launch over a longer period than hitherto announced. Details are yet to be finalized and approved by JTISC, but an overall schedule is planned as follows:

- October 2009 – Commencement of pilot training and first assessor registrations
- December 2009 – Availability of initial public courses, registration and administration services
- January 2010 – First formal release of BPL process set and core scheme documentation
- June 2010 – Target date for initial Foundation grade accreditation
- September 2010 – Target date for start of migration period for current TickIT-registered organizations and auditors
- December 2010 – Target date for Bronze and Silver grade accreditations
- July 2011 – Target date for Gold and Platinum grade accreditations.

It is stressed, however, that these are provisional dates only and may change. Other aspects of the scheme, such as a full documentation set and BPL, will be released at times consistent with this overall schedule. The initial accreditation in June 2010 will likely be for ISO 9001 only, without the planned additional Requirements Standards – essentially ISO/IEC 20000 and ISO/IEC 27001 – which are planned in the second accreditation phase in December 2010. If all goes to plan, it is hoped to have a public seminar promoting the scheme this coming December or January.

Pilot Assessments

Following on from the trials that took place during the development project, it is planned to run, with the principal certification bodies, pilot assessments that can be used to both finalize operational details and also provide accreditation review evidence. If successful, the objective would be to transfer these provisional certificates to accredited status.

Website

As mentioned above, it is hoped to have the **TickIT^{plus}** website properly managed and available as a public source of information within the next month or so. Details such as current delivery schedule, scheme documentation, availability of training courses and planned seminars will be available here, which should be regarded as the first point of call for the latest information on the scheme.

Summary

As always, comments and suggestions are always welcome, either via our website www.tickitplus.org or direct to me on dkirving@iee.org.

Derek Irving,
TickIT^{plus} Development Project Manager



Does Well Engineered Software Deserve a More Systematic Approach to License Management?

by Steve Klos

Overview

Who is responsible for ensuring that software, once installed, can be accurately identified? Depending on your place in the overall Software Asset Management (SAM) ecosystem and your perspective, the answer to this will vary. However, when the question is framed differently – “Who carries the liability if software installed at a customer location is not identified and reported correctly?” – the answer to that question is almost always, “the customer”.

Software purchasers willingly take a risk when they agree to software entitlement terms that allow a software publisher to audit the purchaser’s computing devices to validate they do not have more software installed than they have purchased. If software installations are not tracked and managed properly, it is very easy for software to be over-deployed without the software purchaser even noticing. Software purchasers with a mature software asset management (SAM) program in place can minimize the risks of over-deployment as well as over-purchasing, but there are always trade-offs between the cost of an effective SAM program and the risks taken by a software purchaser.

What are the real costs of inaccurate software identification, and who bears these costs? From the details above, many people would say the software purchaser bears the responsibility for implementing a capable SAM program and any resulting liabilities of not having a SAM program. It is true that the software purchaser needs to implement a SAM program and balance the cost of that program against the risk of liability for not being in

compliance with their software license entitlements. But are SAM programs more costly and complex than they need to be, due to lack of accurate software identification? Are there other organizations in the SAM ecosystem that also bear the costs of inaccurate software identification? This article provides answers to these questions, and also provides details on how ISO/IEC standards currently available or in development will minimize costs for all members of the SAM ecosystem.

It is clear that software identification is not the only area of a SAM program that can significantly benefit from improved accuracy and efficiencies. However, accurate software identification is a relatively straightforward problem to solve, especially since the ISO/IEC standard 19770-2 is expected to be published before the end of 2009. This standard defines a software identification tag structure that will dramatically decrease the costs required to identify software throughout the SAM ecosystem. Other ISO/IEC standards currently under development will further reduce costs of SAM programs as well as other, often unseen, costs that other SAM ecosystem members carry.

Why All the Interest in Software Asset Management?

Interest in Software Asset Management has been growing over the last few years. This interest has been propelled by a number of factors, these include:

- regulation,
- risk management,
- cost savings,
- security management.

Regulation

Many SAM initiatives are driven by stricter government regulations (see details on [Consensus Audit Guidelines](#) (CAG), [Sarbanes-Oxley](#) (SOX), [The UK Combined Code](#), [Health Insurance Portability and Accountability Act](#) (HIPPA), [Electronic Records and Electronic Submissions CFR 21 part 11](#), [Financial Modernization Act of 1999](#) (GLBA), [COBIT](#), and so on). These regulations all have some components that apply to installed software, electronic transactions or corporate governance issues that include software asset management. In fact, using the Google Timeline function, and entering the terms, “regulation software governance”, results in the graph shown in Figure 1 that indicates the frequency of these terms showing up together in press releases:

1980-2009 [Search other dates](#)

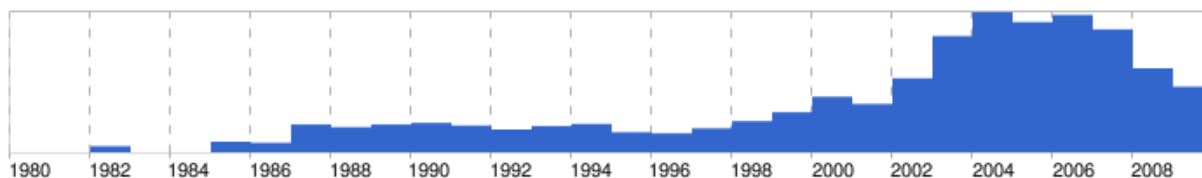


Figure 1: Google Timeline for the Search: “regulation software governance”

Obviously, software governance started to become more interesting after the 1998 UK combined code was introduced, and became even more interesting since 2002 when the Sarbanes-Oxley Act became law.

Risk Management

Adding to the issue of governance is the area of risk management. There are multiple components to risk management, but two of the more prominent public risks regarding software assets are negative publicity from a failed software audit and the fines and unexpected expenses these failed software audits may lead to. Organizations such as the Business Software Alliance (BSA) and the Software and Information Industry Association (SIIA) are part of the SAM ecosystem and have a role to protect and enforce software copyrights. Regardless of each individual’s view of these organizations, most people would agree that any organization purchasing products to run a company (including software products) should adhere to contractual obligations for those products. Not adhering to these obligations when it comes to software may end up causing headlines such as those shown on the next page.

SIIA Reaches Six-Figure Anti-Piracy Settlement with Iowa Corporation

[SIIA press release – March 12, 2009](#) – company is named in press release

Florida-Based Insurance Company Settles With BSA for Unlicensed Software Installations, Agrees to Pay \$70,000

[BSA press release – Sep 2, 2009](#) – company is named in the press release

Cost Savings

In addition to regulations and the risk of negative PR, many organizations are recognizing that by managing their software assets better, they can save money. For example: organizations can realize savings by only renewing the support and maintenance on software actually being used, as well as getting better volume discounts by centralizing purchases.

Security Management

Finally, all organizations – public, private and governmental – are recognizing that there are security implications associated with having software and data stored on a number of different computing devices used throughout their organization. Often, security issues are included in risk management, but security management is becoming an item that requires more individual and specialized attention. Two of the many security issues organizations face are: the requirement to patch software that could otherwise be used to spread mal-ware, and the need to identify software that may open security holes in an otherwise functional firewall system (such as peer-to-peer file sharing systems).

One of the unifying factors for all of these issues is the fact that organizations often cannot effectively manage any of these items unless they have the ability to accurately identify the software installed throughout their organization. If you can't identify it, you can't track it. If you can't track it, you can't measure it. If you can't measure it, you'd better hope it doesn't matter. As we can see from the issues listed above, when it comes to identifying software, it does matter!

Why Are SAM Programs So Hard To Implement?

It should be clear that having an effective SAM program is important for many reasons. For one thing, an effective SAM program can save companies money. So why don't more organizations have an effective SAM program in place today? The short answer is – it's difficult to do well and it can be expensive to implement – there will be savings and cost avoidance, but companies will still have some up-front expenses. Organizations cannot just purchase a SAM tool and be done. To have an effective SAM program in place requires the company to train people, create and validate processes and implement the appropriate technology. Implementing a SAM program requires multiple unique skill sets, as well as an on-going project plan. SAM programs require resources that an organization may not have available, either due to lack of skilled personnel or because the required personnel are dedicated to other projects. Senior management needs to be involved and committed to dedicating the necessary resources to a SAM program in order to ensure an effective program can be implemented and to have it continue to function efficiently.

Once the decision is made to implement a SAM program, a significant effort will be undertaken to create a program appropriate for the organization. A key data element that will occur early in the data collection process is the capture and review of the installed software inventory installed throughout the organization. There are differing methods of capturing software inventory data, but due to the nature of software, the accuracy of the inventory is unlikely to be better than 80% – that is unless other steps in the SAM process are in place to ensure a higher accuracy. Howard Hastings, from CA, has written about the issues of software identification and why it is so difficult to do effectively in a white paper titled, "[Why Software License Management Is So Difficult – and How to Simplify It](#)".

There are many other factors that cause a SAM program to be difficult to implement, including the collection of data related to software entitlements, proof of purchase details, and so on. Often these other data collection efforts, as well as reconciliation efforts, are made much more difficult due to the fact that the

organization's purchase data and the installed software inventory do not correlate with each other. In addition to making reconciliation more difficult, this issue also makes it a more manual and thus resource-intensive and more expensive process.

Why Does Accurate Software Identification Matter?

Accurate software identification is the basis of reporting what is actually installed on computing devices within an organization. Often SAM programs will utilize a third party SAM tool and its propriety methodology of application recognition to determine which applications are actually installed. This process often leads to errors. These errors may be due to:

- **omissions** – the application recognition process does not recognize a software title,
- **errors** – the application recognition process is using incorrect rules to determine a software title,
- **insufficient information** – there are many product structures that cannot be identified accurately with an inventory tool – these include OEM software as well as some suites and bundled products.

Some of these errors can be corrected with other SAM or IT processes (one example of a process that can help solve these problems is locking computers down and only using a desktop management tool to handle software deployment). Some of these can be corrected by adding resources to review and resolve omissions and errors. However, in some cases, it's just not possible to collect information that can be used properly to identify installed software.

The primary issue with software identification is that this is a key data input to a SAM reconciliation process. In an effective SAM program, software inventory of installed software is done on a regular basis and software license entitlements are captured and entered into the SAM tool on a regular basis. In theory, organizations should continually match and reconcile entitlements to the installed software. However, in practice, the reconciliation process is much more problematic. For example: an installed application may have one name, while the actual software entitlement has another name ... and the purchase documents have yet another name. This makes it difficult, if not impossible, to automate what is in effect a three-way reconciliation. The issue of names not matching is very common, especially since there are no standards applied to company names, product names, specific unique reference details for a software title, and so on.

As we'll see later in this paper, starting with an accurate, well-defined product reference with a consistent and registered company reference is a first big step towards resolving many of the costs and difficulties of implementing a SAM program. Please note, this article will not suggest SAM programs are not necessary, only that the cost of implementing and managing the SAM program can be significantly decreased and the process should and will become significantly more automated with appropriate standards.

Who Bears the Costs of Incomplete or Inaccurate Software Identification?

Many SAM ecosystem members consider that the cost of inaccurate software identification is borne by the software purchaser. After all, if an organization's software installation count is found to be over-deployed compared to the number of license entitlements purchased, the software purchaser is liable for additional licensing costs and any fees that may be associated with this issue.

But is that the only SAM ecosystem member that carries a cost associated with incomplete software identification? Let's assume that the software in question comes from a large independent software vendor (ISV) that has multiple departments with different departments created for software development, sales, marketing, finance, support services, and so on. Let's also assume that the software purchaser has a SAM program in place along with a SAM tool to identify installed software, based on its unique proprietary methodology that utilizes unique file signatures discovered during the software inventory. Where do we find the actual inefficiencies and costs of incomplete or inaccurate software identification information?

Software Purchaser

The software purchaser is liable for incorrectly reported information – especially if software is discovered to be over-deployed. However, the costs don't stop there. In many cases a SAM tool will allow end-users to add their own identification information, either to the customer's own database or to the SAM tool's database. This allows end-user organizations to track in-house built applications as well as more unique commercial off the

shelf (COTS) applications. However, as more and more customer specific identification information is entered into the SAM tool, the customer is making the cost of switching SAM tools significantly more expensive. In most cases, application recognition data customers add to a SAM tool's database is done in a proprietary form required by the SAM tool they're using and cannot be transferred to another tool. This implies that the software purchaser will have a higher switching cost built into their SAM program.

SAM Tool Manufacturer

The SAM tool manufacturer must apply resources to creating their application recognition database. This may mean the SAM tool manufacturer has to purchase the software in order to do a clean installation (this will typically result in the most accurate application recognition data), or it may mean they base their work on inventory details provided by one of their customers. There are likely both technical and engineering resources applied to this issue. In addition, the library must be managed and re-distributed on a regular basis incurring more hardware, engineering and networking costs.

Because application recognition is difficult, each SAM tool manufacturer is required to dedicate resources to identify software based on their own unique set of proprietary rules. This result means each SAM tool manufacturer duplicates the same software identification work as the other SAM tool manufacturers in order to provide their customers with software identification.

Ultimately, these costs are borne by the customer, with higher SAM tool purchase or maintenance costs, or with less functionality. Looked at in a different way, SAM tool manufacturers have a lost opportunity cost they could have applied to lower product costs, providing more features, focusing on better quality, and so on. Each one of these lost opportunity costs has the potential to result in fewer SAM tool sales.

Software Publisher

There are a number of costs borne by the software publisher for incomplete or inaccurate software identification – some of these may be hard costs, some are soft costs. We'll review a few examples:

- Sales Teams
 - Software publishers will at times audit the software installed by a customer. If it is difficult for the customer to accurately identify installed software on their own, and they are liable for a significant license fee, the software publisher may be in for protracted negotiations that may require a significant amount of account management effort and likely executives spending time working with the customer. This is time that could be better spent developing new customers or new corporate strategies.
 - Obviously, if disputes between the software publisher and the software purchaser get heated or are protracted, there is a potential the customer may work to find an alternative product. If this happens, the publisher has lost a customer and all the associated product, support, training and consulting revenue that go with that customer.
 - Alternatively, a large number of customers try to stay in compliance with their software entitlement purchases, but do not have the accurate information necessary to ensure they've purchased what they need. By not simplifying the identification process, software publishers may very well be limiting revenue they would otherwise receive.
- Support Teams
 - Organizations that provide support must often deal with customers calling in and not knowing exactly the product, version or patch level of their product. Sometimes this information is easy to find, sometimes it is not. Call times will necessarily include the time required to determine the product the customer has installed.
- Marketing Teams
 - Often marketing teams must develop training for partners' organizations – both those that develop SAM tools and are looking for help in defining the application recognition process, as well as those organizations that provide auditing services. This training can be very extensive, depending on the complexity of the product and on the variety of the entitlements the product is sold under. If there were a reliable, consistent and authoritative method to accurately identify installed software, this training can be significantly shortened and focused on items that are of higher value to the software purchaser.

The above is just a portion of the costs a software publisher is likely to bear due to incomplete or inaccurate software identification. One important thing to realize is that each of these costs is often compartmentalized. Sales teams deal with the audits and difficult customer meetings because they have to – it's done for customer satisfaction. Support teams also deal with software identification because they have to – for customer satisfaction. Training of partners and audit teams has to be done to ensure accuracy.

These costs can generally be reduced, sometimes significantly, by the software publisher providing complete and accurate software identification information.

Who Owns Software Identification Within the Software Publisher?

The owner of accurate software identification should be held within the software development organization. This is the organization that understands exactly what is being installed on a computing device. They also have the specific details to validate the components of a software product and determine how those components are combined during installation to provide certain features. Generally, specific licensing and installation details are defined in coordination with marketing, and then implemented by the development team. Software identification should be part of that definition process.

One issue with this expectation is that there often is very little cost to the development organization if incomplete or inaccurate software identification information is provided. This often means that product development does not have a vested interest in applying a standard that may not gain their engineering teams much value. In these situations, it often takes a manager that understands the full software development and sales cycle to step in, or for engineers who understand the value of accurate information to step forward and promote the concept.

How Can ISO/IEC Standards Resolve These Problems?

By creating and publishing standards that have gone through significant peer review sessions, a common approach is created that will work for a significant number of organizations across multiple industries and across country boundaries. Working Group 21 (WG21), is the group within the ISO organization (shown in Figure 2) that is focused on Software Asset Management standards.

The terms of reference for WG21 are:

- prepare standards to positively impact the Software Asset Management market,
- include industry market requirements in the development of SAM standards,
- support the industry where possible in the uptake of SAM standards.

Scope : ITAM standards used internationally, across any market, any geography and any platform.

Essentially, WG21 is working to promote consistency in SAM processes as well as provide standards that will help lower the costs and raise the value of SAM programs for any company throughout the world.

Review of SAM Standards Published or Currently Under Development

The following provides a summary of the standards WG21 has developed, or is in the process of developing. It is important to note that although each of these standards is identified with the same prefix number, the numbering does not infer a direct association. The work on -1 and -4 are related to each other and are focused on processes, but the -2 and -3 standards are data standards that can be used independently of other 19770 standards.

19770-1 – SAM Processes

Status: Published May 2006

This is a process standard that defines 70 processes grouped into a hierarchical set of categories that are expected to be in place for an organization to be considered having an effective SAM program implemented. The standard simply indicates what the processes are and what should be the outcomes from those processes.

19770-2 – Software Identification Tags

Status: Final vote closed October 20, 2009

Convener: Steve Klos, TagVault.org, Agnitio Advisors Inc.

This is a data standard that defines a set of seven mandatory and 30 optional elements to provide authoritative software identification information for software products. Based on an XML structure, the tagging definition is extensible and also provides support for digital signatures to authenticate provenance and tag integrity.

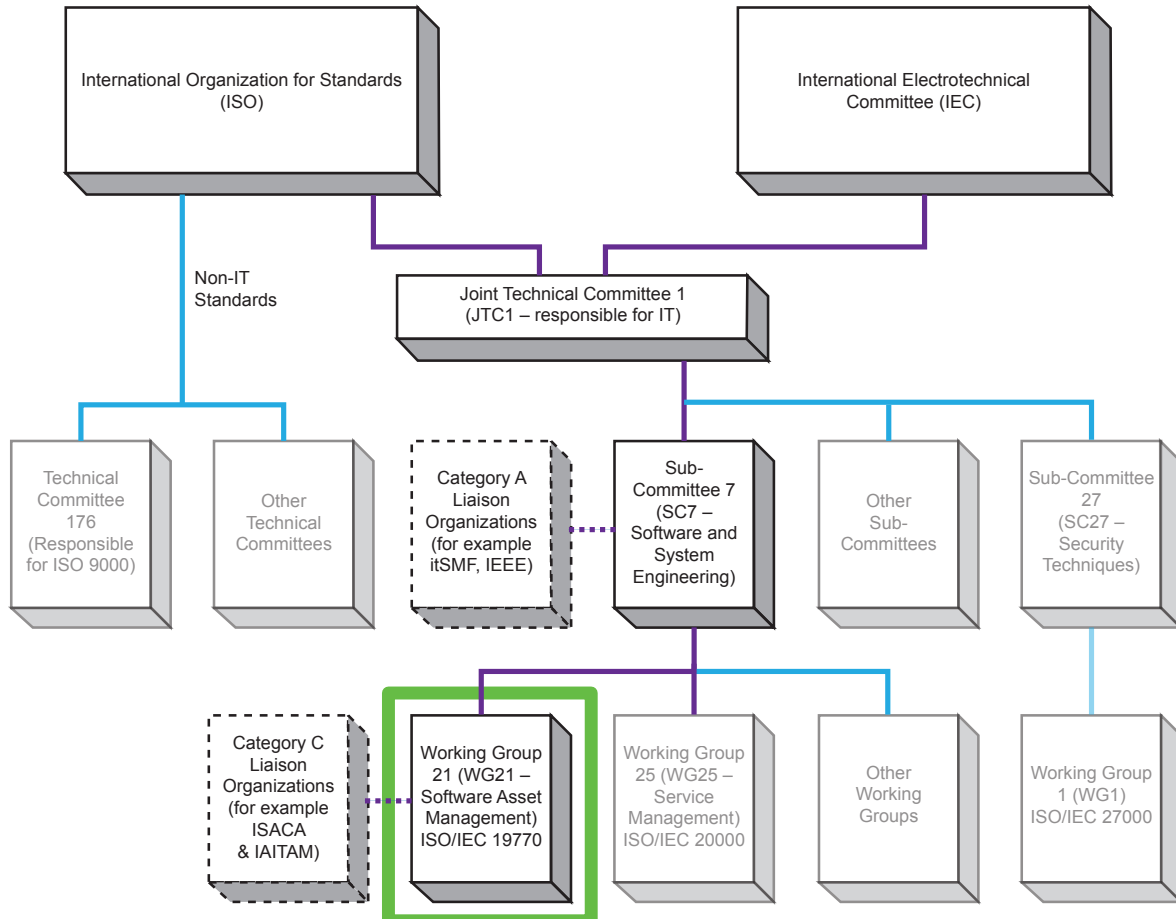


Figure 2: Overview of Organizations Formally Involved in the Standards-Making Process for ISO and ISO/IEC Including for Software Asset Management (image copyright David Bicket 2008, used with permission)

19770-3 – Software Entitlement Tags

Status: Other Working Group (OWG) formed, under development

Convener: John Tomeny, Sassafras Software

This will be a data standard that defines a set of metrics that will be used by enterprise customers to determine if a software entitlement is in use or not. Based on an XML structure, this standard does not use license entitlement terminology to specify an entitlement. Instead, the standard will define the details of what actually needs to be measured to track entitlement usage. This may be as simple as identifying if a software title is installed, all the way to the identification of specific network ports that should be monitored on a server to determine which clients connect and for how long.

19770-4 – Staged Adoption of SAM Processes

Status: Proposed New Work Item

Assigned Convener: David Phillips

This standard will provide an officially sanctioned staging for organizations interested in using 19770-1 processes as a template for SAM program development. This program was originally developed by the BSA

which is assigning the rights to the documented conformance program over to ISO. The methodology is based on a tiered approach to demonstrating adoption of SAM processes, meaning that an organization is expected to have a complete set of processes in place for the assessed tier level and all lower tiers to be considered to be in conformance with the ISO standard at a specified tier level. The tier definitions are based on the hierarchical structure of 19770-1 and are defined on the BSA web site at – <http://www.bsa.org/country/News%20and%20Events/News%20Archives/en/2008/en-11132008-sam-advantage.aspx>.

David Phillips provided an excellent write-up of ISO/IEC 19770-1 in [issue 2Q08 of TickIT](#). He has also provided a detailed summary of the expectations of ISO/IEC 19770-4 in [issue 1Q09 of TickIT](#). The remainder of this article will focus on the details of the ISO/IEC 19770-2 draft standard, as well as provide some information about the certification authority developed as a non-profit organization to bolster market support and product consistency in the rollout of SWID tags.

Resolving the Software Identification Problem – Anatomy of a Data Standard

Development of a software tagging standard was initiated by a clear market demand for more accurate and detailed SAM data being provided to software purchasers. This requirement was quickly split out into two separate standards.

The first standard, ISO/IEC 19770-2, focused on software identification. This standard is related to inventory and discovery processes, and the expectation was that software discovery information would be installed on computing devices along with the software to ensure accurate software identification information is provided to SAM, Desktop Management and Help Desk tools. This became the basis for the development of the ISO/IEC 19770-2 draft standard. Note that as part of the scoping exercise, the working group explicitly excluded the definition of any application locking definition in the SWID file – the tag, as specified, is for identification purposes only.

The second standard – ISO/IEC 19770-3 is more complex, but also potentially more valuable. This standard is related to software entitlements; it provides standardized details on what an entitlement is, along with specific metrics that need to be collected to know if entitlements have been consumed. The expectation for this standard is that the data would follow the same workflow as software purchases; the data for software entitlements would end up with the purchasing department that can then forward it to the IT organization responsible for an organization's SAM program. This became the basis for the development of the ISO/IEC 19770-3 draft standard currently under development.

Usage of the Software Identification Tag

There are quite a number of use cases for the SWID tag and the document for ISO/IEC 19770-2 includes some of these use cases. The three primary interactions with a SWID tag are: creation, modification and consumption. We'll briefly cover the general details of these.

SWID Tag Creation

SWID tags may be created by the creator of the software or by a third party. The SWID tag standard needs to accommodate this and allow anyone using the data from these tags to be able to identify who the creator was. In addition, there is a lag time between a standard being published and the implementation of that standard in products. To accommodate this lag time, and provide value to the market faster, the SWID tag standard allows individuals and software purchasers to create their own SWID tags that provide software identification information. These SWID tags can be used retroactively to apply tags to existing software on existing computing devices. Having a centralized repository where these tags can be shared allows a community-based tagging ability that can be beneficial to all SAM ecosystem members.

SWID Tag Modification

SWID tags are expected to be deployed with software as it's installed. Often, larger organizations manage the installation process through desktop management tools that install and update software using automated means. In these cases, as well as some other use cases, it's beneficial if the desktop management team (or possibly a different third party entity) is able to provide additional details in the SWID tag.

The SWID tag is an XML file and has multiple elements defined, as well as support for extended data to be added. This allows SWID Tag Modification to be supported without much overhead – in fact, an organization can open a SWID tag with a text editor, make the changes they need, and distribute the updated tag.

This use case requires specialized design that ensures that certain data is authentic and has not been modified. Fortunately, the World Wide Web Consortium (W3C) has created a digital signature recommendation for XML files (XMLDSIG) that allows certain elements in the XML file to be digitally signed while other elements can be added, modified, or deleted. Digital signatures add a level of complexity to SWID tags, but they also add a level of authentication to elements that are digitally signed. This paper will not detail the process of digital signatures and how they are applied in this standard, but if the reader is interested, additional information regarding the digital signature process will be available on the TagVault.org web site in the next few months.

SWID Tag Consumption

When discovery tools are run on a computing device, they will typically be the tools that pull in the SWID tags and use the data, either through a centralized server, or to present information on the computing device itself. The SWID tag provides information that can be used to validate that software referenced in the tag is, in fact, installed. The standard defines specified locations where SWID tags should be installed on different platforms.

The standard does not specify how discovered information should be used. This is due to the fact that the discovery agent may be a SAM discovery tool – in which case the tag is used to provide inventory details; it may be an administrative discovery tool – in which case the tag data may be displayed on the desktop, it may be a help desk in which case the tag data is used for inventory for support purposes, or it could be used by any other tool that needs to identify software accurately.

Anatomy of the SWID tag

The SWID tag needs to be something that can be used on any platform, created by any publisher and consumed by multiple different toolsets. An obvious choice for the storage of the data was an XML document and that is the technology the SWID tag is based on. There were additional criteria placed on the structure of the XML definition due to the fact that the W3C XMLDSIG recommendation was put to use for authenticating that data had not changed. Finally, we needed to work through the details of what was the absolute minimum data a SWID tag was required to have, and should a SWID tag be able to be extended. Based on the decision made by the working group, the final structure of a SWID tag included three primary groups of data elements:

- **Mandatory elements** – these data elements must be provided for a tag to be considered valid. This is an absolute minimum set of data that a tag creator needs to provide for SAM, Help Desk and administrative tools to start becoming useful.
- **Optional elements** – these data elements provide either additional data from a publisher that can be used to help identify how a software product may have been installed on the computing device, or may be used by larger organizations in order to track such details as a software release following ITIL release processes.
- **Extended elements** – there will be additional data items that tag creators want to include in a SWID tag. To accommodate this requirement, the standard includes a facility to extend the data elements defined in a specific tag. Note that extended elements may or may not mean anything to SAM tools, but it provides a method that software publishers and/or end-user desktop management organizations can include more details about a software product in a standardized fashion.

A sample of a SWID tag is shown in Example 1.

Providing Mandatory Tag Elements

There are seven mandatory elements that must be included in every SWID tag for the tag to be considered valid. These elements were the absolute minimum set of data the working group determined would be required for the effective use of the data and that software publishers would readily have available and be able to provide.

```

<swid:software_identification_tag
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#" xmlns:swid="http://standards.iso.org/iso/19770-2/2008/schema.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://standards.iso.org/iso/19770-2/2008/schema.xsd
software_identification_tag.xsd ">

  <!-- Mandatory elements -->
  <swid:entitlement_required_indicator>>false</swid:entitlement_required_indicator>
  <swid:product_title>Firefox</swid:product_title>
  <swid:product_version>
    <swid:name>2.0.0.10</swid:name>
    <swid:numeric>
      <swid:major>2</swid:major>
      <swid:minor>0</swid:minor>
      <swid:build>0</swid:build>
      <swid:review>10</swid:review>
    </swid:numeric>
  </swid:product_version>
  <swid:software_creator>
    <swid:name>Mozilla Corporation</swid:name>
    <swid:regid>regid.1994-11.com.mozilla</swid:regid>
  </swid:software_creator>
  <swid:software_licensor>
    <swid:name>Mozilla Corporation</swid:name>
    <swid:regid>regid.1994-11.com.mozilla</swid:regid>
  </swid:software_licensor>
  <swid:software_id>
    <swid:unique_id>firefox.2.0.0.10</swid:unique_id>
    <swid:tag_creator_regid>regid.1994-11.com.mozilla</swid:tag_creator_regid>
  </swid:software_id>
  <swid:tag_creator>
    <swid:name>Mozilla Corporation</swid:name>
    <swid:regid>regid.1994-11.com.mozilla</swid:regid>
  </swid:tag_creator>

  <!-- Optional elements -->
  <swid:abstract lang="en">Corporation's flagship browser.</swid:abstract>
  <swid:data_source>Electronic distribution</swid:data_source>
  <swid:license_linkage>
    <swid:activation_status>Fully licensed</swid:activation_status>
    <swid:channel_type>Internet</swid:channel_type>
    <swid:customer_type>End-user</swid:customer_type>
  </swid:license_linkage>
  <swid:product_category>
    <swid:UNSPSC_ver>10.0501</swid:UNSPSC_ver>
    <swid:segment_title>Information Technology Broadcasting and
Telecommunications</swid:segment_title>
    <swid:family_title>Software</swid:family_title>
    <swid:class_title>Network applications software</swid:class_title>
    <swid:commodity_title>Internet browser software</swid:commodity_title>
    <swid:code>43232705</swid:code>
  </swid:product_category>
</swid:software_identification_tag>

```

Example 1: Sample ISO/IEC 19770-2 Software Identification Tag as presented in the ISO/IEC 19770-2 document in Annex H.

The seven elements are:

- **Product Title** – this data element is straightforward – it provides the title that an end-user would typically see displayed on splash screens and about screens. It should also be data that's used consistently on software purchase orders as well as on software license entitlement documentation. In general, this element will not be used for automated processing, but is rather a human-friendly name for the product and will be used in reports.
- **Product Version** – this data element provides two different entries:
 - The first entry is a string value that contains the version detail as the end-user would see it. This version string can be any mix of alphanumeric and is not required to follow a specific format. This allows vendors that utilize specialized version numbers that they have trained their users to utilize with the ability to continue to include this information in the SWID tag
 - The second entry is a set of four numeric values. This data will be used to allow automated sorting and grouping of various versions of the product.
- **Software Creator Identity** – This element provides information about the organization that created the software. There will be instances where a third party creates a SWID tag and this element helps to differentiate the fact that the creator of the software did not create the SWID tag. See **Regid** below for more information on how this data is structured for consistency.
- **Software Licensor Identity** – This element provides information about the organization that owns the licensing rights to the software. There will be instances where one organization created the software and a different organization owns the licensing rights to the software. See **Regid** below for more information on how this data is structured for consistency.
- **Software Unique Identifier** – This is a unique identifier that, when combined with the tag creator identity, must be globally unique. Providing a completely unique identifier ensures that future standards (such as the ISO/IEC 19770-3 draft on software entitlements) have the ability to reference a specific unique reference for a particular product if required. The standard does not require that this element utilize a Globally Unique Identifier (GUID) such as the Microsoft defined implementation of Universally Unique Identifiers (though it can if the tag creator desires), but rather it specifies that the tag creator ensures that every combination of tag creator + unique identifier is unique for every software product, or software patch that's distributed with a SWID tag.
- **Tag Creator Identity** – This element provides information about the organization that created the tag. There will be instances where a third party creates a SWID tag and this element provides differentiation between the tag creator and the software creator or the software licensor. See **Regid** below for more information on how this data is structured for consistency.

Regids

One of the issues the working group had to deal with was that there is no normalization applied to company names. In fact, with today's inventory systems, organizations will often have multiple different ways to reference the same software publisher. For example, IBM could be referenced as IBM, I.B.M., International Business Machines, Intl' Business Machines, and so on. This is not to mention the misspellings of a company name. To provide a more consistent and repeatable approach to identifying a specific publisher, the working group used the Internet Small Computer System Interface (iSCSI) standard as an example, and defined the regid element.

The regid is a structured definition that is based on the domain of an organization along with the date the domain was owned by that organization (note that there are rules specifying how the month is determined for the regid – these rules ensure that domain transfers can be identified using the regid). The structure also provides for additional entities within a publisher to be included, if required, to expand the namespaces used for unique references (see Software Unique Identifier above).

A regid structure is shown in Example 2.

Utilizing the structure, all the regid references to IBM, for example, would be referenced as:

Regid.1986-04.com.ibm

Even if there are groups within IBM that are included as separate naming authorities (that is, they manage their own Unique ID references), reporting systems can still automatically group all IBM regids into one group

Type	Date	Naming Auth	Additional "example.com" naming authority
regid.1995-09.com.example,		Applications	
regid.1995-09.com.example,		Utilities	

Example 2: Examples of a Regid

simply by utilizing the first three components of the regid. Additionally, if a third party is creating SWID tags, the rules used to create a regid can be applied without any requirement to create a new registration, filing paperwork or other procedures that often delay efforts to use the 'correct' information.

Providing Optional Tag Elements

There are 30 optional elements defined in the SWID tag. Many of these elements are designed to assist an individual who may be looking for purchasing or entitlement details for a specific product installation. For example, the License Linkage element, if provided, may specify if a particular software installation came from media that was intended to go to a governmental organization, for training purposes, for volume resale, or for retail distribution.

Other optional elements are designed to support the ITIL framework by providing sign-off information that can be retained with a software product as that product goes through the various testing, pilot and production phases.

One very interesting component of the optional elements is the Package Footprint element. This element provides support for a vendor to include specific file references and other details of a software product. By providing this information, the tag can be used to provide information to discovery engines that detail how to identify software that does not have tags. This element can also provide significantly more security to an organization. For example, if a publisher includes all the files for their software product in the Package Footprint and those details are digitally signed by a trusted agent, organizations can now validate both the provenance of the application as well as provenance of specific files related to that application. A system that does a full file inventory as well as collecting the tags can now identify all trusted files and can also immediately highlight any files that are not trusted.

Extending the SWID Tag definition

The working group realized that with a defined structure applied to software identification, software publishers and end-users would likely want to incorporate additional information into the SWID tag that may not be pre-defined. The SWID standard allows organizations to create their own schemas and add their own data elements into the SWID tag.

Bringing the SWID Tag Standard to Market

The ISO/IEC 19770-2 draft standard provides a structured approach to providing information that can be used for multiple purposes in an organization (help desk, software asset management, administrative details, and so on). When creating this standard, the requirement included as part of the remit was that the standard should not require a registration authority to be useful. Significant efforts had to be made to ensure this requirement was met. The standard has been created so anyone familiar with XML can create, distribute, modify or use a SWID tag just by reference to the standard document.

However as the working group defined various details, there were elements that were necessarily left more flexible in the definition and did not have explicit values defined. These include information such as activation status, channel type, customer type, and so on. These elements will need to be developed for a multitude of markets, publishers and platforms.

It would, however, be very helpful if an organization were available to help with the development and documentation of these types of elements to ensure some level of consistency and normalization. Fortunately, Symantec, CA and ModusLink OCS see value in having an organization like this as well, so they have worked with IEEE-ISTO to create a non-profit organization – TagVault.org to provide tools and services in this area.

Make It Easy

The basis of creating an XML file is fairly straightforward and many engineers could use the [schema definition \(XSD\) of the ISO/IEC 19770-2 draft](#) to create XML files in their sleep. But the skills to create an XML file are not universally known, and some development organizations would prefer to save time in the creation, validation and management of tags. This is something the non-profit organization helps with by providing its members with tag creation and signing tools.

One area that becomes more complex is the area of digitally signing a SWID tag. The process to sign a tag, as well as the structures required to create a signed tag that includes, for example, a timestamp, are not well documented by the W3C XMLDSIG recommendation. These are areas where the tag creation and signing tool can provide very significant value even to highly trained engineering departments.

Provide Support

TagVault.org is providing details on what is required to have a certified SWID tag, how digital signing should be applied to a tag, how to validate digital signatures, and so on. All this information is provided to members, and much of it is provided to any engineer who's interested. TagVault.org provides services to its members, but it also provides value to its members by educating the market on appropriate approaches to creating, distributing, modifying and using SWID tags.

Ensure Compatibility

Even when standards are developed, there are often many different ways to approach a specific requirement, no matter how tightly defined it is. TagVault.org is providing documentation, reference implementations, member-submitted source code, tools and services to ensure that all SAM ecosystem members utilize SWID tags in the same way.

Provide Added Value

TagVault.org is also providing services that add value to SWID tags. One example is that the organization is developing a repository that will be used to store certified tags, as well as provide a storage resource for members to share tags that are used for software identification purposes. Imagine being able to define the software identification details for one application recognition program, and then having those details transfer to a new application recognition program. Although this will lessen the ability for SAM tools to lock in their customer base, the support of the repository ends up being a positive thing for the application recognition programs due to the widely diverse set of users who can now create and share tags.

SWID Tags Benefiting The Whole SAM Ecosystem

The ISO/IEC 19770-2 draft detailing the structure of software identification tags was created for use by all members of the SAM ecosystem. The benefits start with the software publisher and continue all the way through the distribution, sales, consulting and support market infrastructure, with the software purchaser ending up with the ability to accurately identify software installed in their computing environment.

When SWID tags are created by a software publisher, they will likely be created by the development team. The unique thing about SWID tags is that the development team probably gets the least benefit from these tags – after all, the software developers have a very rich versioning system in their source code control environment. This means that support for the tags will need to be fostered by engineers who understand the full implications of the SWID tag and the value it brings to the market.

Once a software product is distributed from the publisher, the fact that the product contains an authoritative SWID tag will reduce costs for all members of the supply and support chain that the product travels through. This includes the cost for SAM tools, since there will no longer be a need to create unique proprietary identification libraries that are likely to become out of date as soon as a patch is released (with software tagging, each patch will carry its own SWID tag providing further capabilities to the discovery tool).

The software publisher will receive real cost benefits due to the fact that most of the items detailed in the section – *'Who bears the costs of incomplete or inaccurate software identification?'* are minimized. Additionally, the software publisher can receive significant public relations benefits because, for the first time, the software

publisher is telling the user community exactly how to identify software. The software purchaser will recognize the additional transparency provided by those publishers who provide software tags.

The entire SAM ecosystem benefits from the consistent use of SWID tags, but the software purchaser is the ultimate benefactor. Organizations have learned to fear an audit or a license review from a software publisher (or their agent). There are multiple reasons for this fear, but a common one is that organizations have little idea of the potentially high costs that may come from an audit or license review. Customers do not understand these potential costs because they do not have a SAM program in place, likely due to the fact that SAM programs are difficult and potentially expensive to implement. The ISO/IEC 19770-2 draft is the first step towards making SAM programs easier and less costly to implement and maintain, as well as providing a significantly higher degree of inventory accuracy. Once the draft is published, many in the industry expect that customers will require their software suppliers to include SWID tags in their software products as part of the contract negotiation process.

Get Involved – Promote SWID Tagging

Standards help the market the most when they are applied consistently across the industry to which they apply. The ISO/IEC 19770-2 standard is nearing completion and is getting close to publication (estimated in late 2009). Once the standard is available, engineers should be promoting the benefits of the standard and the low costs of implementation. If a software publisher analyzes all the costs associated with the fact that their customers cannot accurately identify what they have installed – from support costs, to license review negotiations, to lost sales – the payback for including tag data is likely one release of one product.

The public relations benefits of getting involved with the SWID tagging standard early will also promote customer goodwill that comes from the recognition by the customer that their publisher is actively working to be part of the solution to a protracted issue that has existed since software started to be sold.

Finally, if a software publisher expects to implement SWID tags in the near future, there is significant benefit to that publisher's getting involved with a non-profit organization tasked with making that process easier, less expensive and more consistently applied. TagVault.org is providing registration services for SWID tags – this ensures that the tags meet not only the conformance requirements as specified in the standard, but that it meets market-driven requirements for consistency of application. As a member organization, TagVault.org also provides tools and services to benefit its membership. The first tool available from the organization is one that allows for the automated creation, validation and digital signing of SWID tags. If this is of interest to your organization, contact TagVault.org at info@tagvault.org or go to the website at www.tagvault.org.

Steve Klos is the convener the ISO/IEC 19770-2 draft focused on software identification. Based on his work on the software identification tag standard, Steve became the executive director of TagVault.org – the certification authority for 19770-2 software identification tags and the organization providing tools, services and specialized knowledge for software identification tags to the SAM ecosystem. Steve co-founded Agnitio Advisors, a company focused on developing, evaluating and improving software asset management policies and procedures. He also co-founded ManageSoft Corporation, a provider of desktop, security and software asset management tools and services. Steve is the recipient of multiple industry certifications in Software Asset Management and has been privileged to become a Fellow of IAITAM – one of the largest and most respected Asset Management industry associations. Steve is a certified instructor for SAM training classes and has presented at major conferences such as IAITAM, Gartner, CA World and SoftSummit.



Stop Press: see next page for update to this article

Update

The ISO/IEC 19770-2 final draft international standard (FDIS) draft vote has come in – the draft submitted was unanimously approved. What does this mean? It means that the software market now has a standard definition for software identification based on an ISO standard.

A number of people volunteered time in the development of this standard. A list of individuals in the other working group who volunteered their time to develop this standard is available for review. These folks all deserve recognition for a job well done! Note that it will take a bit of time for the ISO editorial team to finalize the published version of the standard – once that's done, a copy of the standard can be purchased from your local standards organization (ISO, ANSI, BSI, Standards Australia, and so on), or even from Amazon.com. The standard is currently expected to be available before the end of 2009.

A major benefit of this standard is for software purchasers when negotiating with software vendors - the expectation is that software purchasers will negotiate the requirement for software identification (SWID) tags into their purchase agreements.

Software Quality Unpeeled

by Dr Jeffrey Voas

The expression '*software quality*' has many interpretations and meanings. In this article, I do not attempt to select any one in particular, but instead help the reader see the underlying considerations that underscore software quality. Software quality is a lot more than standards, metrics, models, testing, and so on. This article digs into the mystique behind this elusive area.

The term software quality has been one of the most overused, misused, and overloaded terms in software engineering. It is a generic term that suggests quality software but lacks general consensus on meaning. Attempts have been made to define it. The Institute for Electronics and Electrical Engineers (IEEE) Standard 729 defines it as: ... *totality of features of a software product that bears on its ability to satisfy given needs and ... composite characteristics of software that determine the degree to which the software in use will meet the expectations of the customer.* [1]

However, this attempt and others are few, and not precise. In fact the second edition of the Encyclopedia of Software Engineering [2] does not have it listed as an entry; the encyclopedia skips straight from "Software Productivity Consortium" to "software reading." And worse, books with software quality in the title never give a definition to it in their pages [3, 4].

If you review the past 20 years or so, you will find an abundance of other terms that have been employed as pseudo-synonyms for software quality. Examples include process improvement, software testing, quality management, the International Organization for Standardization 9001, software metrics, software reliability, quality modelling, configuration management, Capability Maturity Model Integration, benchmarking, and so on. In doing so, the term *software quality* has wound up representing a family of processes and ideas more than it represents *good enough* software. In short, software quality has become a culture and community more than a technical goal [5].

In this article, I will avoid the quicksand associated with trying to come up with a one-size-fits-all definition. Instead, I will expose how software quality is composed of various layers and, by peeling off different layers, it allows us to have a rational discussion between a typical software supplier and end user such that an agreement can be reached as to whether or not the software is good enough.

Certification

We will begin dissecting software quality by first looking at the multiple viewpoints behind the term *certification*. This will provide us with a look into our first layer.

The term is often used to refer to certifying people skills. For example, the American Society for Quality (ASQ) has a host of certifications that individuals can attain in order to demonstrate competence in certain fields, for example, they can become an ASQ Certified Software Quality Engineer. An individual can also become certified in specific commercial software packages, for example, a Microsoft Certified Software Engineer.

For the purposes here, I employ a different perspective that comes from three schools of thought. The first school deals with certifying that a certain set of development, testing, or other processes applied during the pre-release phases of the life cycle were satisfied. In doing so, you certify that the *processes were followed and completed*. (Demonstrating that they were applied correctly is a trickier issue.) In the second school, you certify that the developed software *meets the functional requirements*; this can be accomplished via various types of testing or other analyses. For the third school, you can certify that the software itself is *fit for purpose*. This third school will be the most useful, and throughout this article, it will be considered for software to be *good enough* if it is fit for purpose.

In this article, the term *purpose* suggests that two things are present: (1) executable software and (2) an operating environment. An *environment* is a complex entity: it involves the set of inputs that the software will receive during execution, along with the probability that the events will occur ^[6]. This is referred to as the *operational profile* ^[6]. But it also involves the hardware that the software operates on, the operating system, available memory, disk space, drivers, and other background processes that are potentially competing for hardware resources, and so on. These other factors are as much a part of the environment as are the traditional inputs; they have been termed *invisible* or *phantom* users.

In some instances phantom users more heavily determine whether the software is fit for purpose than the traditional inputs. In short, it is environment that gives fit for purpose context. By more completely defining and thus bounding the environment to include phantom users, we gain an advantage in that we can reduce the set of assumptions needed to predict whether the software is good enough. Understanding the distinction between traditional inputs and phantom users is one ingredient needed to argue that *fit for purpose* has been achieved.

Further, note that rarely will there be only one environment that software, and in particular general purpose software, will encounter during operation. That offers a key insight as to why general purpose software is not certified by independent laboratories; such laboratories could not be omnipotent and could not know all of the potential target environments ^[7].

By revisiting the three schools of thought on certification, we discover eight ways to visualize software quality (see Table 1). Let us look at a couple.

Scenario	Meets Requirements	Satisfies Development Processes	Fit for Purpose
1	No	No	No
2	No	No	Yes
3	No	Yes	No
4	No	Yes	Yes
5	Yes	No	No
6	Yes	No	Yes
7	Yes	Yes	No
8	Yes	Yes	Yes

Table 1: Views on Certification

In Table 1, scenario 2 represents a system that did not meet the requirements and was not developed according to the specified development procedures but, miraculously, the end result was software that was usable in the field. While this seems implausible, it is possible. Scenario 7 is the opposite: a system that met

the requirements and was developed according to specified development procedures but resulted in unusable software. Scenario 7 may seem like heresy to many in the community of software quality practitioners. It is not; it simply dispels the myth that requirements elicitation is far from a perfect science and that simply following common sense *do's* and *dont's* (as spelled out in a development process plan) guarantees good enough software [8].

Note that only four of these scenarios yield good enough software: 2, 4, 6, and 8. The other four provide a product that is not usable for its target environment and that brings us back to the discussion of why scoping the target environment as precisely as possible is an important piece of what software quality means.

In summary, *fit for purpose* is the nearest of the three certification schools of thought of the IEEE definitions for software quality. However, we cannot only rely on knowing the environment and expect to be justified in proclaiming we have achieved software quality. Let us explore other considerations.

Three High-Level Attributes of Fit for Purpose

Most readers would probably be comfortable with labelling software as being of good quality if the software could ensure that (1) it produces accurate and reliable output, (2) it produces the needed output in a timely manner, and (3) it produces the output in a secure and private manner. These three criteria simply state that you get the right results at the right time at the correct level of security. These are the next three considerations that cannot be ignored and must be incorporated into what software quality means.

While each of these is intuitive, none is precise enough. The family of attributes referred to as the '*ilities*' is a good starting point to help increase that precision [9]. This family includes behavioral characteristics such as reliability, performance, safety, security, availability, fault-tolerance, and so on. (These attributes are also sometimes termed non-functional requirements.) Other family members such as dependability, survivability, sustainability, testability, interoperability, and scalability each require some degree of one or more of the first six attributes. So, for example, to have a dependable system, some level of reliable and fault-tolerant behavior is necessary. To have a survivable system, some amount of fault tolerance and availability is required, and so on. However, to simplify this discussion towards our goal of understanding the term software quality, we will focus only on the first six: reliability, performance, safety, security, availability, and fault-tolerance.

"Ility" Oxymorons

Table 2 illustrates combinations of these six *ilities*. If we were to flesh this table out as we did in Table 1 we would have 64 rows; however, here we only show 14 combinations for brevity. (For the cells left empty, we are not considering the degree to which that attribute contributes to the quality of the software's behavior.)

Category	Reliability	Performance	Safety	Security	Availability	Fault-tolerance
1	Yes	Yes	Yes	Yes	Yes	Yes
2	Yes		No			
3	No		Yes			
4	No			Yes		
5	No				Yes	
6		No			Yes	
7		Yes	No			
8			Yes	No		
9			No	Yes		
10	Yes			No		
11	Yes			No	Yes	
12	Yes	No		Yes		
13		Yes		Yes		
14	No	No	No	No	No	No

Table 2: *Ility* Combinations

Let us look at a few of these categories and determine what they represent:

- **Category 1:** Suggests that the software is reliable, has good performance, does not trigger unsafe events to occur (for example, in a transportation control system), has appropriate levels of security built in, has good availability and thus does not suffer from frequent failures resulting in downtime, and is resilient to internal failures (that is, fault tolerant). Is all of this possible in a single software system?
- **Category 2:** Suggests that the software offers reliable behavior, but suffers from the likelihood of producing outputs that send the system that the software feeds inputs into an unsafe mode. This would represent a safety-critical system where hazardous failures are unacceptable; hazardous failures are categorized differently for such systems than failures that do not facilitate possible disastrous loss-of-life or loss-of-property consequences. (Note that software by itself is never unsafe; however software is often referred to as unsafe if it produces outputs to a system that put the system into an unsafe mode. Safety is a system property, not a software property.) A classic example of a reliable product that is unsafe is placing a functioning toaster into a bathtub of water with the cord still connected; the toaster is reliable, but it is not safe to go near.
- **Category 3:** Suggests that the software behaves so unreliably when executed that it cannot put the system into an unsafe mode. An example here would be that the software gets hung up in a loop and the safety functionality is never invoked.
- **Category 8:** Suggests safe but not secure software behavior. This is quite realistic for a safety-critical system with no security concerns. Note that the interesting aspect of this category is how safety and security are defined. Many people use these terms interchangeably, which is incorrect.
- **Category 11:** Suggests that the software behaves reliably and has good availability, but lacks adequate security precautions. Many systems suffer from this problem.
- **Category 12:** Suggests that the software behaves reliably, is extremely slow, but has adequate security. It makes one wonder if the system is so slow that it is effectively unusable, and thus secure, since it would take too long to break in.
- **Category 13:** Suggests high levels of security and high levels of performance. In certain situations that is plausible, however typically security kills performance and vice versa.
- **Category 14:** This is the easiest combination to achieve. Anyone can build a useless system.

The main point here is that the aforementioned high-level attributes (1) produce accurate and reliable output, (2) produce the needed output in a timely manner, and (3) produce the output in a secure and private manner are actually composed of the lower-level *ilities*. Another important point not to overlook is the fact that some of the *ilities* are not compatible with one another. An example of this can easily be found using fault tolerance and testability. A final important point is that some combinations of the *ilities* are simply counterintuitive, such as a system that is safe but unreliable.

One last thing to note: it is vital to get solid definitions for the *ilities* and to know which ones are quantifiable. For example, reliability and performance are quantifiable; security and safety are not. This makes it far easier to make statements such as *we have very high reliability but an unknown level of security*.

The Shall Nots

There is yet another layer in the notion of fit for use that deals with *negative* functional requirements. Think of a negative requirement as “the software shall *not* do X,” as opposed to a functional requirement stating that “the software *shall* do X.”

Negative requirements are far more difficult to elicit than regular requirements. Why? Because humans are not programmed to anticipate and enumerate all of the bad circumstances that can pop up and that we need protection against; we are instead programmed to think about the *good* things we want the software to do.

For certain types of systems, particularly safety-critical systems, enumerating negative requirements is a necessity. And for software requiring security capabilities, security rules and policies are its equivalent to negative requirements. For example, a negative security requirement could be that the software shall never open access to a particular channel unless it can be guaranteed that the information passing through the channel is moving between trusted entities. The difficulty in defining *shall not's* for security is that we cannot imagine all of the

different forms of malicious attacks that are being invented on-the-fly, and if we cannot imagine those attacks, we likely will not prevent them.

Before leaving the topic of negative functional requirements, it is worth mentioning an interesting relationship between them and the environment. So far, we have only mentioned traditional inputs and phantom users as players in the environment. Traditional inputs are those that the software expects to receive during operation. But there are two other types of inputs worth mentioning: *malicious illegal* and *non-malicious illegal*. A malicious illegal input is one that someone deliberately feeds into the software to attack a system, and a non-malicious illegal input is simply an input that the system designers do not want the software to accept but has no malicious intent. In both cases, filtering on either type of input can be useful to ensure that certain inputs do not become a part of the environment and in doing so ensure that negative functional requirements are enforced.

Time

The next layer in our quest for software quality is *time*. Software has fixed longevity; it can be expanded, as we learned from Y2K, but not indefinitely.

One of the easiest ways to explain why time fits here is to look at the situation where a software package operates correctly on Monday but does not operate correctly on Tuesday. Further, the software package was not modified between these days. (This is the classic problem that quickly carves down the number of freshman computer science majors.) Why has this problem occurred?

It all goes back to the importance of environment in the understanding of software quality. Earlier we defined the environment as inputs with probabilities of selection, hardware configurations, access to memory, operating systems, attached databases, and whether other background processes were over-indulging in resources, and so on.

But what is not mentioned was *calendar time*. Environment is also a function of time. As time moves forward, other pieces of the environment change. And so while all effort and expense can be levied toward what we perceive is evidence supporting the claim that we have good enough software, we need to recognize that even if we do have good enough software, it may be only for a short window of time. Thus, software quality is *time-dependent*, a bitter pill to swallow.

Cost

We cannot end this article without mentioning *cost*. The costs associated with software quality are exasperated by the un-family like behaviors of various *ilities*. Not only are there technical trade-offs discovered when trying to increase the degree to which one *ility* exists only to find that another is automatically decreased, but there is the financial trade-off quagmire concerning how to allocate financial resources between distinct *ilities*. If you overspend on one, there may not be enough funds for another. And, as if the technical considerations are not hard enough when trying to define software quality, the financial considerations come aboard, making the problem worse.

Conclusion

In this article, a set of layers for what software quality means has been unpeeled. I have argued that a more useful perspective for what software quality represents starts from the notion of the software being *fit for purpose*, which requires:

1. Understanding the relationship between the *functional requirements* and the *environment*.
2. Understanding the three high-level attributes of software quality: the software: (a) produces *accurate and reliable* output, (b) produces the needed output in a *timely* manner, and (c) produces the output in a *secure and private* manner.
3. Understanding that the *ilities* afford the potential to have varying degrees of the high-level attributes.
4. Understanding that the *shall-not* functional requirements are often of equal importance to the functional requirements.

5. Understanding that there is a *temporal* component to software quality; software quality is not static or stagnant,
6. Understanding that the *ilities* offer technical incompatibilities as well as financial incompatibilities.
7. Understanding that the environment contains many more parameters such as the phantom users than is typically considered.

Thus, software quality, when viewed with these different considerations, becomes a far more interesting topic, and one that will continue to perplex us for decades to come.

References

1. IEEE. *Standard Glossary of Software Engineering Terminology* IEEE. American National Standards Institute/IEEE Standard 729-1983: 1983.
2. Marciniak, J., ed. *Encyclopedia of Software Engineering*. Second Edition. Wiley Inter-Science: 2002.
3. Wiecek, Martin, and Dirk Meyerhoff, editors. *Software Quality: State of the Art in Management, Testing, and Tools*. Springer. New York: 2001.
4. Gao, J.Z., H.S. Jacob Tsao, and Y. Wu. *Testing and Quality Assurance for Component-Based Software*. Artech House. Norwood, MA: 2003.
5. Whittaker, J., and J. Voas "50 Years of Software: Key Principles for Quality." IEEE IT Professional. 4(6): 28-35, Nov. 2002.
6. Musa, John D., Anthony Iannino, and Kazuhiro Okumoto. *Software Reliability: Measurement, Prediction, Application*. McGraw-Hill, NY, 1987.
7. Voas, J. "Software Certification Laboratories?" Apr. 1998.
8. Voas, J. "Can Clean Pipes Produce Dirty Water?" *IEEE Software* July 1997.
9. Voas, J. "Software's Secret Sauce: The 'Iilities.'" *IEEE Software* Nov. 2004.

Jeffrey Voas, Ph.D., is currently director of systems assurance at Science Applications International Corporation (SAIC). He was the president of the IEEE Reliability Society from 2003-2005, and currently serves on the Board of Governors of the IEEE Computer Society. Voas has published numerous articles over the past 20 years, and is best known for his work in software testing, reliability, and metrics. He has a doctorate in computer science from the College of William & Mary. E-mail: j.voas@ieee.org



DIY WORKBOOKS
save time and money

ISO 9001 TickIT
ISO 9001 RAD
ISO 9001 SSADM
ISO 9001 PRINCE
ISO 9001 Service Delivery
Listed on the Internet

http://www.nvo.com/management_systems-author

E. Sutherland, IEE Ind Aff
Trog Associates Ltd
P.O. Box 243
UK - South Croydon/Surrey, CR2 6NZ.
Tel +44 (0) 208 786 7094
Fax +44 (0)208 686 3580
Email trog@dial.pipex.com

ISO 9000/BS7799
Guaranteed Results!

- 100% clients registered first time
- Consultant/Trainer since 1990
- Lead TickIT Auditor since 1992
- TEC and Business Link funded projects
- Practical advice and training

Consultancy Training Pre-Assessments

Bal Matu BSc(Hons) C.Eng.MIIEE FIQA MAQMC
Lead TickIT Auditor
Tel/Fax: 44 (0)1928 723701
bal@bsmq.demon.co.uk

