A Generic Framework for Automated Quality Assurance of Software Models Supporting Languages of Multiple Paradigms

Darryl Owens and Mark Anderson
Department of Computing, Edge Hill University, Ormskirk, Lancashire
Email: darryl.owens@edgehill.ac.uk

***Abstract*—Software Quality Assurance (QA) is a key area in the development and maintenance of scientific software systems in order to ensure the reliability of the output generated by such systems. Approaches taken in implementing QA within the lifecycle include manual techniques, which require developer intervention, and automated techniques, which can be completed by analysis toolsets. Manual QA techniques are labour intensive and time-consuming to complete. This paper highlights the main areas of software quality assurance and assesses the area in terms of tools that exist to automate these techniques. These tools are evaluated at a high level to allow general statements to be made and the key issue of non-generic tools that are applied across multiple language paradigms. Reviewing the background of automated software quality assurance and general software quality assurance. A framework is then proposed to fill the gap in automated software quality assurance, with the proposal to develop this framework.**

***Index Terms*—software quality assurance, software testing, automated software engineering, programming language paradigms**

I. Introduction

Quality Assurance (QA) can be seen to cover a wide area, thereby there lies a need to define QA within the context of this work. The focus of this paper is to consider the implementation of software quality assurance techniques which are automated rather than requiring end user intervention. The reasoning underpinning this is driven by the reduction in workload that can be brought about through the automation of such techniques. The figure below shows a structure in which software testing and quality assurance is relative to overall quality assurance engineering, which itself is a field within software engineering [1].

Opinion is divided in realtion to the application of quality assurance techniques within the software development lifecycle. A traditional view is that this is a practice that takes the form of testing at the end of development and is a independent part of software engineering [2]. As quality of software has increasingly become a necessity of scientific software development, growing opinion has identified the need to apply quality assurance techniques, such as testing techniques, throughout the developmental lifecycle [3].

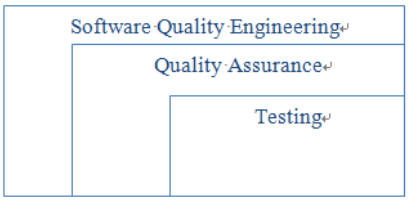

Figure 1. Scope and content hierarchy [1]

A definition for quality software is said to be a piece of software that meets or exceeds a customers specification when considering functionality, performance, reliability, availability and supportability at a cost less than or equal what the customer expects to pay [4]. Rather then define software quality Krutz et al [5] states that "There is a lack of commonly agreed-upon definitions for software quality, but it is possible to refer to software quality by its common attributes", which is also supported by the work of Tian [1]. The important feature for software quality are identified as usability, efficiency, maintainability and portability and less important are performance, availability and supportability. However, contradictory views would suggest that functionality, performance, reliability, availability and supportability are the critical deterministic values when considering the quality of software [4]. The former definition aligns well with the ISO-9126 standard. However, it is acknowledged that other frameworks exist and that ISO-9126 does not completely cover all areas of software quality [1]. An example of such a framework is "CUPRIMDS (capability, usability, performance, reliability, installation, maintenance, documentation and service)" [1] used by IBM in the software development lifecycle. It has also been identified that "many companies and communities associated with different application domains have adapted and customised existing quality frameworks to define" [1]. An example of such a community is BITS financial services who have developed a software quality assurance framework for financial institutions focusing on software security with context specific key area such as; IT risk controls embedded within core business processes; techniques, practices, and tools that identify security vulnerabilities;

Manuscript received October 12, 2012; revised March 7, 2013.

doi:10.4304/jsw.8.9.2191-2196

integrating software from third parties; and investment in the development of resilient software components [6].

Although there are many opinions surrounding quality assurance standards, the fundamental conceptual underpinnings of the ideals are the same. Quality is a measurement based on end user expectations. However, it is the criteria to be measured which are a cause for discussion. The areas that are listed for each standard or definition of QA can be generalized and the range of areas is based on several factors including client, industry and software purpose. For example a software product made for the financial sector retains a significant focus on security and may require high levels of usability. Alternatively, a software application which batch processes files without user intervention yet still designed for the financial sector will retain the security measure but will not be reliant upon the usability. This would appear to demonstrate that for software to be determined to be of quality then the metrics to be adopted must be determined from the outset of its design and development.

We can therefore derive three critical areas of QA on which this paper shall focus; functionality, reliability and maintainability can be extracted from ISO-9126 and driven by software testing that is related to quality assurance. Usability will not be considered as this paper is focused on the batch processing elements of software systems. Furthermore, it has been determined that" projects pertaining to the scientific area differ in their quality assurance and testing process compared to other organizations" [1][7].

II. Static and Dynamic Analysis

There are two categories of analysis that shall be considered when looking at software testing; static and dynamic testing. Static analysis is the evaluation of a code base without that code base being executed [8]. This may be undertaken by automated toolsets. Static analysis allows for identification of such potential issues as memory corruption errors, buffer overruns, out-of- bound array accesses, or null pointer de-references [9]. There exists a body of evidence which supports static analysis as an effective tool in the QA process [10], and there are many examples of static analysis in use [11], [12], [13], [14], [15], [16]. Static analysis has been used in all areas of software development. A number of tools have been developed to automate this process currently in use [17], [18], [19], [20]

Unlike static analysis, dynamic testing makes use of test plans, execution of test cases and evaluation of results [21]. This technique can be used to run functional, logical, interface and bottom-up tests as well as others [21]. Dynamic analysis has the advantage of generating more detailed information, as it doesn't rely on abstract program states [20]-[22]. There exist examples of dynamic analysis being applied in different situations [16]-[23], however static analysis is more broadly adopted as a technique in industry as the tools used for dynamic analysis are relatively uncommon [17], [20], [24]

There are significant advantages to using both types of analysis, however, to create a more comprehensive tool for quality assurance. It has been suggested that more than one type of analysis must be used [23][25] for this purpose. The research discussed in this paper shall implement both static and dynamic analysis to allow the scope of the research to cover a larger area of issues within automated quality assurance.

III. Analysis of Testing and Tools

Prior to analyzing the tools that are available for deployment in the quality assurance of software, and evaluation of testing methods and techniques must first be performed. Subsequent evaluation of the tools will consider those used to automate some of these QA processes to establish those of greatest significance to this work.

A. Testing Methods

Testing methods are the means by which the testing will be completed. Multiple methods are usually used and some testing objectives are focused on specific methods. However multiple methods can usually be applied to each type of testing. Testing can generally be considered to fall into one of two major categories:

Black Box/Functional-External behavior is observed for correctness during execution via the software input and output [1].

White-Box/Structural-Verifies the implementation of internal parts of the software; has been done correctly e.g. data structures, statements of code etc. [1].

It can be inferred that IBM link dynamic analysis with black box testing, and also static analysis with white box testing [26]. This contradicts the afore-mentioned definitions; white and black box are testing methods, looking at software as only inputs and outputs or viewing the structure, whereas static and dynamic analysis are used to find issues via one or more of the methods. For example, dynamic analysis may use the black box method. However it could also use the white box method to analyse software.

B. Testing Levels

Testing levels are used to describe where the testing should be taking place. Each test could be run at different levels but much like the testing methods some objectives are specifically targeted at a certain level of testing. Examples of levels include

Unit/Component-Software is broken down into units, a single cohesive function or procedure and tested separately [27]

Integration-When separate pieces of pre-tested code are place together these are then tested for the correct results [27]

System-The software is tested as if it were in use; the data inputted is what data users would be expected to input [27]

Acceptance-Run by the client, the software is tested against set criteria to see if it meets the clients needs [27]

C. Objectives of Testing

The objective of the test is to identify what is being tested. The objective is usually combined with a level and

a method to make a test plan. For example, a beta test is targeted at acceptance level as the product is being delivered for execution on the end user system. Beta testing can also be considered a black box test as the user won't have access to the internal structure and code, and will be analyzing the test results based on input and output alone.

Compatibility testing-tests regarding information sharing with other software e.g. copying text from a web page to a office document [28]

Regression tests-Upon the correction of a fault, all areas that are affected or linked with the changed code should be re-tested to prevent the introduction of new faults [27]

Stress testing-running software with lower than intended specification e.g. slow CPU, lower memory, etc. [28]

Load testing-contradicting stress testing, load testing attempts to push the software to its limits and example of this is giving it as much data as it can handle [28]

Alpha testing-distribution of a few copies of the software to key individuals or clients to test what has been developed to that date. [28]

Beta testing-Tested as a full product by external entities most likely potential users, that will use the software as expected to unearth faults and provide feedback based on their experience [27]

Usability testing-based in ergonomics this is the testing of having someone interact with the software (usually based on standards and guidelines) [28]

Accessibility-technically under usability testing, accessibility looks at disabled users or users with impairments and how these individuals can use the software. E.g. visual impairments [28]

Internationalization and localization-testing that software can be used in different geographic areas this could mean taking into account language, local conventions etc. [28]

Code coverage-testing for unreachable code. Code that, no matter what circumstances the software is in will never run. Analyzers can be used to give you measurements of how much code is used [28]

Release testing-High-level checks to make sure the software does as documented all exportable versions are up to date and all files are present [27]

IV. Analysis Tools

There are a number of toolkits available for testing software applications. These are targeted at the automation of testing to remove the load on the developer and/or end user when testing the code base. The tools will generally focus on specific aspects of software testing, or will only be applicable for code developed in a single language or paradigm. The following section will discuss the key tools which are available to developers and consider the strengths and limitations of each.

Two tools which are closely aligned in terms of their scope, and the scope of this paper, are FPT [17] and Malpas [15]. These tools are used within the scientific industry, and assess large programs for quality. Malpas, however, only uses static analysis. From this, it could be deduced that FPT covers a wider variety of programming issues and bugs as FPT uses both static and dynamic analysis. However, one of the key goals of Malpas is to support the quality assurance analysis of safety critical systems [15]. It might therefore be considered that Malpas would need to adhere to specific standards whereas the use of FPT is much broader within the scientific community. Both tools are, however, similar in that they adopt techniques to create a degree of language independence; FPT uses a internal representation to analyse FORTRAN code and Malpas uses its Intermediate Language to analyse Ada, C and Pascal. This is a key observation in relation to this paper as a framework is being designed to support multiple paradigms. Whilst the languages which can be analysed are imperative, the concept of separating the language from its analysis is critical.

A further static analysis tool which has been applied to the development of scientific models is Polyspace [19]. Polyspace, like Malpas, adheres to development/industrial standards. However Polyspace is specifically designed for use with embedded systems. Unlike Malpas, which uses an intermediate language to create some language independency, Polyspace is embedded into specific IDEs and therefore does not create language independence. It does, however, provide a suite of programs which can be applied to a variety of programming languages. This would lead to the belief that the conceptual techniques which are being applied can be ported to a range of language paradigms, but the implementation of those techniques is language dependent.

In terms of identifying tools which support both the dynamic and static analysis of code, then an alternative to FPT lies with JNuke [20]. JNuke uses its 'general' analysis (a combination of static and dynamic analysis) to construct more robust and accurate tests. Like FPT, JNuke can be used to analyse code developed in a single language. However as JNuke uses no language independency, unlike FPT which uses an internal representation, JNuke is completely interlinked with the language which it can analyse (Java in this case) as it makes use of a very novel approach using a customized JVM to implement additional features (such as Backtracking). The developers of JNuke have created their own VM written entirely in C to implement the capabilities used to analyse Java code in greater depth.

In comparison, TestingAnywhere [29] and Cantata++ [24] are very similar as both focus on using GUI input to facilitate the automation of tests with minimal user intervention. TestingAnywhere utilizes a unique "SMART" tool which records macros and allows the user to edit these in order to change values of tests. Cantata++ takes an alternative approach and, whilst supporting the use of a GUI for test configuration, also implements white box testing whereas TestingAnywhere would appear to implement sequences of black box tests to achieve a similar outcome.

V. Evaluation of Tools

The tools which have been considered in the preceding section would, between them, suggest that the critical features for a generic framework can be implemented. Functionality, reliability and maintainability, as identified in section 1, can be achieved if a deep level of testing and analysis can be performed. For this to be successful then the analysis tools must support a range levels, objectives and methods. The tools should also support the simple configuration of these tests and facilitate the automation of testing.

When evaluating the toolsets considered, it is also important to consider the language paradigms that are supported by them as these may have an affect of the methods used to test the software. A generic framework must be able to support the analysis of code irrespective of the language or paradigm that has been adopted for its development. To this extent, there are four key paradigms that must be addressed [30]. It is acknowledged that some languages can be used to develop software utilizing different paradigms dependent upon the requirements of the model being developed. For example, it is possible to develop object-oriented code or procedural code using *C++*. For the purposes of this study, each language will be aligned with the paradigm, which closely matches it specification.

When taking this into account, the analysis tools themselves are then closely related to the paradigms of the languages that they support. For example, JNuke [20] and Parasoft [18] would be closely aligned to object-orientation in supporting Java and *C++*. Furthermore, Malpas, Polyspace and FPT [15][17][19] are aligned to procedural languages such as Ada, FORTRAN and C. In this respect, Cantata++ [24] would appear to be an exception as the languages that it addresses are both procedural and object-oriented. However, the scope of testing within Cantata++ is limited to unit testing that will not significantly affect testing between these two paradigms.

VI. A Framework for Automating Software Quality Assurance

The goal of this work is to design a framework for software quality assurance testing which is language independent. The research aims to overcome issues that have been faced in previous attempt to derive a generic framework for this purpose by developing customized techniques based upon related work which has been undertaken [31].

One such attempt to develop a language independent framework made use of an intermediate language. A excellent example of a QA tool adopting this approach is Malpas [15]. There are other cases not linked to QA which utilize such a framework. These would include .NET, in which each .NET language is converted into MSIL (Microsoft Intermediate Language) before being compiled [32]. The inherent limitation of this approach is that the intermediate language must, itself, be developed to adopt a programming paradigm and the notion of language independence is therefore removed.

A further mechanism for implementing language independence is the use of an internal representation. This method has been used within the design of automated QA toolsets such as FPT. In this case the internal representation is not used to implement language independence, but rather is used to remove any language specific issues and allow the QA techniques to be implemented independently. However, it is postulated by the authors that this technique could be adapted to support language independence within a framework.

An issue with an internal representation is the process of parsing a language to be represented within an internal representation. This may be implemented by developing a parser for each language. However, this would be impractical due to the extensive nature of syntax and semantics adopted by modern high-level languages. An alternative solution would be the creation of a meta-language, which is "… a notation for defining the syntax of a language by use of a number of rules" [33]. Examples of such meta-languages are BNF and EBNF which are seen to be de-facto standards for this representation [34]. Alternatives include 'van Wijngaarden grammar' which is a two level grammar, similar to using a meta language, and was originally devised to define the syntax of ALGOL 68 [35].

Using a meta-language to describe the programming language syntax has a further advantage in relation to QA; overcoming the identified issue that "even such a clear and well-designed languages as Pascal contained hidden semantic irregularities which were revealed only by formalization of its semantics" [36].

The research will adopt a meta-language to develop the framework in the first instance, allowing the identification of semantic errors as an additional feature, which enables languages to be described then described in an internal representation for analysis. Should errors be found following analysis then the meta-language could be utilised to convert the edited internal representation back into its original source code. It is proposed that BNF be used at the outset of the work, as it is likely that there will already be a BNF description of most languages [33].

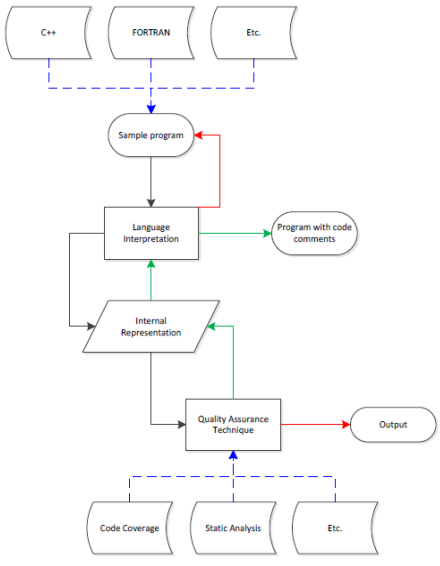

Figure 2. Framework definition

The framework has then to implement this language independence into a form in which QA techniques can be applied. Fig. 2 is a representation of the framework and the processing which will be supported in the analysis of software models.

VII. Conclusions

This paper has presented an overview of software quality assurance within the context of software engineering related to scientific model. Tools which have been developed for QA, specifically those used to automate QA techniques, have been evaluated and the key elements which these tools support have been identified. Most significantly, it can be observed that the tools available are developed either with a focus on specific application areas, or to support specific language paradigms. However, the techniques adopted by these tools to implement QA analysis can be extrapolated to extend the range of coverage that the toolsets currently address. This could be through the support for multiple language paradigms, or for a broader range of applications. Whilst some tools offer support for a wider range of language paradigms, for example through intermediate languages and internal representations, the implementation of these tools restricts the full potential being achieved. The aim of the project, which is discussed in this paper, is to develop a generic paper for QA analysis, designing and assessing techniques that are currently in use to automate software quality assurance. An output of this work will be the development of a taxonomy of QA procedures and techniques which will be based on language paradigm enabling a correlation to be drawn the techniques adopted and the paradigms supported. The goal is to inform the design of a generic framework which, when implemented, can be applied across multiple language paradigms.

Acknowledgment

The authors would like to thank John Collins and Brian Farrimond of SimCon Ltd., and David Gill of UCAR for their support in the development of this project work.

References

[1] J. Tian, "Software quality engineering: Testing quality assurance and quantifiable improvement," *IEEE Computer Society*, Hoboken, 2005.

[2] R.Yin and X. M. Ding, How to improve the quality of software testing. *International Conference on Systems and Informatics*, 2012, pp. 2533-2536.

[3] L. Rosenberg, "Software quality assurance engineering at NASA," *Aerospace Conference Proceedings*, 2002, pp. 2569-2575.

[4] D. P. Wesenberg and K. Vansaun, "A system approach for software quality assurance," in *Proc. IEEE National Aerospace and Electronics Conference*, 1991, pp. 771-776.

[5] R. L. Krutz, R. D. Vines, and G. Brunette, "Cloud security: A comprehensive guide to secure cloud computing," Hoboken: Wiley, 2010.

[6] BITS. (23 October 2012). Software Assurance Framework. [Online]. Available: http://www.bits.org/publications/security/BITSSoftwareAssurance0112.pdf

[7] G .U. Maheswari and V. V. R. Prasad, "Optimized software quality assurance model for testing scientific software," *International Journal of Computer Applications*, vol. 36, no. 7, 2011.

[8] A. Austin and L. Williams, "One technique is not enough: a comparison of vulnerability discovery techniques," *International Symposium on Empirical Software Engineering and Measurement*, 2011, pp. 97-106.

[9] D. Bell and P. G. Brat, "Automated software verification & validation: An emerging approach for ground operations," *IEEE Aerospace Conference*, 2008, pp. 1-8.

[10] B. Chess and C. Wysopal, "Software quality assurance for the masses," *Security & Privacy, IEEE*. vol. 10, no. 3, pp. 14-15, 2012.

[11] N. Truong, P. Roe, and P. Bancroft, "Static analysis of students' java programs," in *Proc. Sixth Australasian Conferance on Computing*, 2004, vol. 30, pp. 317-325.

[12] G. Naumovich, G. Avrunin, L. Clarke, and L. Osterweil, "Applying Static analysis to software architectures," *Software Engineering-ESEC/FSE*, M. Jazayeri and H. Schauer, eds. Springer Berlin / Heidelberg, 1997, pp. 77-93.

[13] T. Reps, T. Lev-Ami, M. Sagiv, and R. Wilhelm, "Putting static analysis to work for verification: A case study," in *Proc. ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2000, pp. 26-38.

[14] N. Ward, "Code verification with the aid of Malpas," *IEE Colloquium on High Integrity Ada*,1999, pp. 3/1-3/3.

[15] K. Harrison, "Static code analysis on the C-130J hercules safety-critical software," *UK Iternational Systems Safety Conference*, 1999.

[16] D. Balzarotti, et al. "Saner: Composing Static and Dynamic Analysis to Validate Sanitization in Web Applications," *IEEE Symposium on Security and Privacy*, 2008, pp. 387-401.

[17] SimCon. (1995). Sim Con-Fortran Analysis, Engineering & Migration. [Online]. Available: http://www.simconglobal.com/

[18] IBM. (2011). IBM Rational AppScan: Application security and risk Managment. [Online]. Available: http://www.sebyde.nl/uploads/media/IBM_Rational_Appscan_family_Data_Sheet.pdf

[19] MathWorks. (1994). Static Analysis with Polyspace Products. [Online]. Available: http://www.mathworks.co.uk/products/polyspace/

[20] C. Artho, et al. "JNuke: Efficient dynamic analysis for java,"in *Proc 16th International Conference Computer Aided Verification Lecture Notes in Computer Science*, USA: Boston, MA, 2004.

[21] R. Fairley, "Tutorial: Static analysis and dynamic testing of computer software," *Computer*, vol. 11, no. 4, pp. 14-23, 1978.

[22] A. Biere and C. Artho, "Combined static and dynamic analysis," in *Proc. Intl. Workshop on Abstract Interpretation of Object-Oriented Languages*, 2005.

[23] M. Salah, S. Mancoridis, G. Antoniol, and M. D. Penta, "Scenario-driven dynamic analysis for comprehending large software systems," in *Proc. 10th European Conference on Software Maintenance and Reengineering*, 2006, pp. 80-90.

[24] Q. Systems. (n.d.) *Cantata* - The Unit Testing Tool for C/C++. [Online]. Available: http://www.qa-systems.com/cantata.html

[25] W. E. Wong, "An integrated solution for creating dependable software," *Computer Software and Applications Conference*, 2000, pp. 269-270.

[26] IBM. (2011). IBM Rational AppScan: Application security and Risk Managment. [Online]. Available: http://www.sebyde.nl/uploads/media/IBM_Rational_Appscan_family_Data_Sheet.pdf
[27] C. Britton and J. Doake, "Testing and handing over the system," *Software System Development : A Gentle Introduction*, K. Reade, K. Mosman, A. Duijser, and J. Bishop, Eds. 4th ed. Maidenhead: MCGraw-Hill Education, 2006, pp. 175-180.
[28] R. Patterson, *Software Testing*, 2nd ed. Indianapolis, lnd.: Sams Publishing, 2006.
[29] A. Anywhere, (n.d.) TestingAnywhere. [Online]. Available: http://www.automationanywhere.com/Testing/
[30] A. Laird, "The four major programming paradigms topic paper #17," *Computer Science*, April 3, 2009.
[31] J. Collins, B. Farrimond, D. Flower, M. Anderson, and D. Gill, "Removal of numerical drift from scientific models: A case study using WRF," *Accepted for Publication in the International Journal of Software Engineering and Applications*.
[32] MSDN. (n.d.). Compiling to MSIL. From Microsoft Developer Network. [Online]. Available: http://msdn.microsoft.com/en-us/library/c5tkafs1(v=vs.71).aspx
[33] ISO. (1996). ISO/IEC 14977: 1996(E). [Online]. Available: http://www.cl.cam.ac.uk/~mgk25/iso-14977.pdf
[34] L. M. Garshol. (2005). BNF and EBNF: What are They and How do They Work? [Online]. Available: http://www.garshol.priv.no/download/text/bnf.html
[35] B. Edupuganty and B. Bryant, "Two-level grammar as a functional programming language," *The Computer Journal*. vol. 32, no. 1, pp. 36-44, 1989.
[36] D. A. Watt and O. L. Madsen, "Extended attribute grammars," *The Computer Journal*. vol. 26, no. 2, pp. 142-153, 1983.

Darryl Owens was born in Yorkshire, UK in 1991. Mr Owens was awarded a BSc (Hons) in Computing at Edge Hill University in the UK in 2012. He is a PhD student in the Department of Computing at Edge Hill University. His PhD study focuses on Quality Assurance within Software Engineering and he is involved with the Software Quality Assurance Research Group, where is looking to develop a generic framework to facilitate the automation of quality assurance across languages of disparate paradigms.

Dr Mark Anderson was born in Liverpool, UK in 1971. Dr. Anderson was awarded a BSc (Hons) in Computer Science by the University of Liverpool, UK in 1993. Mark gained a PhD at the Department of Computer Science in University of Liverpool in 1997.

He has been employed as a Senior Lecturer at Edge Hill University, UK and has led courses at both undergraduate and postgraduate levels. He has also successfully supervised a number of students through the dissertation/project process and is currently the Programme Leader for BSc(Hons) Computing (Application Development) course. He is currently leading the international Software Validation Project on behalf of the department. The project consists of an expanding team and is currently investigating quality assurance of one of the most significant software models in the world. He is also lead the department's involvement on the international HistorySpace project, and is working with national organisations in developing further projects.

Dr. Anderson is a Chartered IT Professional with the British Computer Society, and is a Fellow of the Higher Education Academy.