Experiences and Advances in Software Quality
Monograph: Experiences and Advances in Software Quality  
(published jointly with Novática*)  
Guest Editors: Darren Dalcher and Luis Fernández-Sanz

2 Presentation: Improving Quality in Business Processes, Products and Organizational Systems — Darren Dalcher and Luis Fernández-Sanz

6 Preventative Software Quality Control: Using Human Checking to Change Defective Human Practice — Tom Gilb and Lindsey Brodie

14 The Software Process Improvement Hype Cycle — Miklós Biró

21 Quality Going for Gold — Derek Irving and Margaret Ross

26 Can Teamwork Management Help in Software Quality and Process Improvement? — Esperança Amenguá-Alcover and Antònia Mas-Picacho

34 Evidence-based Software Engineering and Systematic Literature Reviews — Barbara Kitchenham, David Budgen, and O. Pearl Breton

42 Software Project Success: Moving Beyond Failure — Darren Dalcher

51 Software Measurement for Better Project and Process Quality — Christof Ebert


67 A Quality Evaluation Model for Web2.0 e-Learning Systems — Stephanos Mavromoustakos and Katerina Papanikolaou

UPNET (UPGRADE European NETwork)

75 From Mondo Digitale (AICA, Italy)  
History of Computing  
The Turing Test: History and Significance — Giuseppe O. Longo

CEPIS NEWS

89 Selected CEPIS News — Fiona Fanning

* This monograph will be also published in Spanish (full version printed; summary, abstracts, and some articles online) by Novática, journal of the Spanish CEPIS society ATI (Asociación de Técnicos de Informática) at <http://www.ati.es/novatica>.
Improving Quality in Business Processes, Products and Organizational Systems

Darren Dalcher and Luis Fernández-Sanz

Quality is intangible, yet it has been in the forefront of software development and software engineering for well over thirty years. During this period the perception of quality has changed from an activity that takes place downstream, to a management constraint that defines the success of the ultimate product and, beyond that, to a set of quality standards and improvement approaches. Experience has taught us that quality cannot be engineered into the later stages of development and must instead be purposefully designed into the process, product and organizational systems. Consequently, many organizations search for new ways to improve their business processes, increase their maturity profile, and enhance the efficiency and effectiveness of their software development practices.

This special issue brings together a collection of reflections and experiences from leading experts in the field of software quality. Many of the papers report on new ideas and advances thereby offering novel perspectives and approaches for improving quality in software. The papers are grounded in both research and practice and therefore deliver insights that summarize the state of the discipline while indicating avenues for improvement and placing new trends in the context of improving quality in an organizational setting.

The nine papers selected for the issue showcase three perspectives in terms of the trends identified within the software quality domain. The first three papers report on new initiatives and the continuing evolution and improvement of older ideas (mainly in relation to process improvement). The next four papers introduce new ways of thinking and working in practice, while the final two papers feature new technologies and their impact on quality.

Inspections offer a well-established formal review mechanism originally developed for use on the source code of a program or a portion of a program as a systematic procedure for the detection of defects. The review is conducted by peers who attempt to look at the document from a wider perspective with the aim of uncovering lapses that may have

The Guest Editors

Darren Dalcher is a Professor of Software Project Management at Middlesex University and Director of the National Centre for Project Management. He has been named by the Association for Project Management as one of the top 10 "movers and shapers" in project management in 2008 and has also been voted Project Magazine’s Academic of the Year for his contribution in "integrating and weaving academic work with practice". Following industrial and consultancy experience in managing IT projects, Professor Dalcher gained his PhD in Software Engineering from King’s College, University of London. In 1992, he founded and has continued as chair of the Forensics Working Group of the IEEE Technical Committee on the Engineering of Computer-Based Systems, an international group of academic and industrial participants formed to share information and develop expertise in project and system failure and recovery. He has written over 150 papers and book chapters on software engineering and project management. He is Editor-in-Chief of Software Process Improvement and Practice, an international journal focusing on capability, maturity, growth and improvement; editor of a major new book series, Advances in Project Management, which synthesizes leading edge knowledge, skills, insights and reflections in project and programme management and of a new companion series Fundamentals of Project Management which provides the essential grounding in key areas of project management. He is a Fellow of the Association for Project Management and the British Computer Society, and a Member of the Project Management Institute, the Academy of Management, the Institute for Electrical and Electronics Engineers, and the Association for Computing Machinery. He is a Chartered IT Practitioner. <d.dalcher@mdx.ac.uk>.

Luis Fernández Sanz is a Graduate (Universidad Politécnica de Madrid, 1989) and Doctor (Universidad del País Vasco, 1997) in Computer Science. He is currently lecturing in the Computer Sciences Department of the Universidad de Alcalá. In his 20 years of experience in education and research he has also lectured at the Universidad Politécnica de Madrid and the Universidad Europea de Madrid, where he was head of department and director of degree certification. He has combined this work with consulting and project management work for a number of entities. He has authored numerous books and articles at a national and international level. He is a member of the General Board of Directors of ATI (Asociación de Técnicos de Informática, the Spanish Informatics Society) and is a coordinator of its Software Quality group, as well as being the editor of the journal REICIS (<www.ati.es/reics>). <luis.fernandezs@uah.es>. 
eluded the author. While there are benefits associated with peer reviews, the process is tedious, demanding and extremely expensive. The paper by Gibb and Brodie makes a case for moving the inspection process upstream to uncover the error density earlier in the process and for sampling quality levels rather than spending time fixing the defects. Sampling facilitates more informed decision making, while emphasizing the need for quality standards and thus results in defect prevention and appreciation of the need to eliminate errors. Moreover, the results of defect analysis, which offer a short term improvement, also reveal where and how defects occur and can therefore be used as the basis for process improvement. Preventive software inspections as introduced in this paper represent a departure from accepted practices with a real potential to improve quality processes and culture within organizations.

One of the more practical streams within software quality practice has concentrated on the idea of software process improvement (SPI), and has spawned numerous maturity models and capability frameworks. SPI advocates the systematic improvement of software processes by assessing software processes against process standards and frameworks and by mapping levels of achievement. The paper by Biró surveys the historical development of process improvement and its impact on the software practice using the hype cycle as a lens. The adoption of CMM by the US Department of Defense has accelerated the rate of adoption of SPI, giving additional legitimacy to this burgeoning area. The paper maps additional developments such as ISO 9000, Bootstrap, ISO/IEC 15504, explaining their role in the cycle associated with the SPI movement. The ideas of maturity and capability have been exported to other domains and disciplines but have also been criticized for their bureaucracy and inflexibility. Additional perspectives such as the one offered by agile methods may provide a new ground for the coming together of SPI methods and approaches.

The most common SPI approaches (CMMi and ISO 15504) have been implemented in a wide variety of environments and sectors. Consequently, a considerable number of companies and institutions have amassed practical experience of their strengths and weaknesses. Although the common methods have evolved over time, they seem to have become more stable. Other SPI approaches continue to evolve to address existing and future challenges. This indeed is the case of one of the traditional references in the area of software quality: the United Kingdom’s TickIT, a quality system combined with a certification scheme. Irving and Ross present the new scheme for TickITPlus to be launched in 2010. The new framework aims to cover the various needs of organizations and industry. The paper “Quality: Going for Gold” is a good example of how an article can offer readers useful information regarding a forthcoming trend.

However tempting the idea may be, we simply cannot rely on new methods and processes to solve all our problems. A key concern in quality, whether in the software development domain or more generally, has always revolved around the human factor. Moreover, software development efforts tend to be human resource intensive. Although people are clearly central to development, not much rigorous and practical research has been carried out to explore the implications for software quality. "Can Teamwork Management Help in Software Quality and Process Improvement?" represents one clear contribution to exploring the topic. The work by Mas and Amengual addresses one of the most critical items for software development: effective and efficient teamwork. The relationship between teamwork and SPI methods is also analysed in the paper.

Evidence-based practice traces its roots to evidence-based medicine and its concern with a process of systematically finding, appraising and utilizing findings as the basis for clinical decisions. It has been adopted in many domains and disciplines primarily due to its success in unlocking vast data resources which inform decisions and provide a wider and better informed basis for identifying effective remedies for individual cases. Kitchenham, Budgen and Brereton have been instrumental in importing the idea of providing evidence from research integrated with practical experience and human values to improve decision making to the field of software engineering. Systematic literature reviews aggregate empirical results in a methodical way. They have the power to overturn "common knowledge" and to uncover additional evidence which may point to alternative explanations and practices. Many decisions made in software engineering suffer from insufficient data. The approaches proposed in this paper encourage practitioners to consider the scientific evidence underpinning their decisions and can pave the way towards improving the basis for making decisions. In the longer term it may lead to the development of a body of empirical evidence that can be used to improve decision making in software practice.

Various studies suggest that a large proportion of software development projects fail either fully or partially. The normal measure in such studies is the ability to meet initial targets for cost, time and performance. The paper by Dalcher argues for a move beyond simplistic failure studies which are predicated on internal project management efficiency criteria. Indeed, the fascination with failure needs to be replaced with a healthier interest in what is required to build success. In practice, success extends beyond such simplistic internal measures. Many of the issues identified in following the analysis of failure and success stories are to do with relationship management, politics, trust, expectations and escalation; factors that are not included in the typical failure surveys. A wider perspective with multiple representations of the levels of success is developed which expands the time horizons and looks at effectiveness, rather than efficiency, and the outcomes of a project. Effectiveness re-establishes quality as a primary consideration in determining the success of a project (thereby supplanting the delivery to pre-defined budget and schedule as the ultimate criteria). Success is a complex and multi-layered concept. By re-framing our focus of interest from failure to success we may finally be able to progress the discussion...
Experiences and Advances in Software Quality

about success track records in delivering software.

As we have seen software development can be confusing, sending us mixed messages. Given this messiness, how do we know what we need to measure? The failure or success of projects, achievement of quality objectives... How can one know what is happening in each case? Deming’s statement “In God we trust, all others bring data” is a good expression of we mere mortals’ need for measurement in trying to make sense of the environment. Traditionally, the software development community has been reluctant to compile comprehensive and systematic collections of data. Ebbert, the author of a recent book, offers a good review of the discipline, covering foundations as well as projects and process measurement. One of the strengths of this paper is the linking of metrics to SPI and project performance with practical examples.

Although, as shown in the previous papers, the software quality discipline embraces a considerable variety of approaches, methods and techniques, the field is also highly dependent on the evolution of software engineering. As we have seen throughout the emergence of earlier development paradigms (e.g. object orientation, UML, web engineering, etc.), software quality needs to engage, respond and react when new scenarios arise. One of the recent trends in software engineering is the service orientation of software. Following its emergence into mainstream, SOA (Service Oriented Architecture) is pushing the advancement of quality methods for software engineering. The team from the Universidad de Oviedo (García-Fanjul, Palacios, Tuya, and De la Riva) analyses the challenges of testing software services compositions overcoming limitations of traditional testing approaches with practical consequences for BPEL (de facto SOA standard in industry).

But it is not only specific advances in software engineering methods or approaches that drive the evolution of the software quality discipline. New types of software and systems also require adaptation and evolution of traditional methods (or even the creation of new ones) to address their specific characteristics. Mavromoustakos and Papanikolaou have accepted the challenge of contributing to software quality evaluation of systems with a widespread presence in our daily life: E-learning systems with web 2.0 capabilities. Their detailed work provides a comprehensive review of the many aspects that need to be taken into account for an evaluation with the support of a commonly accepted standard such as ISO 9126.

The initial interest in software quality was triggered by concerns about the performance of software and the perception of poor success rates associated with software development projects. Many advances have been proposed over the years and the practice has improved and become better established. As we have seen software development remains a challenging occupation which forces software quality experts to invent new methods and approaches, develop new perspectives and respond to new technologies. While the silver bullet is still missing, it is interesting to observe the vitality and diversity that continue to define and revitalize the area. In this issue, leading researchers and practitioners have surveyed the development of ideas, perspectives and concepts within software quality and given us a glimpse of some of the potential solutions. There is still a long way to go but the journey now seems more exciting.
Useful References about Software Quality

The following references, along with those included in the articles this monograph consists of, will help our readers to dig deeper into this field.

Books

Articles and Papers

Journal Special Issues Dedicated to Quality Topics
- Software Testing, a special issue of IEEE Software containing a number of papers on testing practices, and agile software testing, IEEE Software23(4), July/August 2006
- Quality Requirements, a special issue of IEEE Software containing a number of articles on quality requirements as the front end of attaining quality, IEEE Software, 25(2), March/April 2008.

Web Sites
Preventative Software Quality Control: Using Human Checking to Change Defective Human Practice

Tom Gilb and Lindsey Brodie

Traditional Software Inspection is often uneconomic and ties up valuable staff resources. Shifting the emphasis from cleanup (that is, from identifying defects and then removing them), to merely sampling the major defect level of specifications, produces significant benefits. It enables the quality level of specifications to be determined more rapidly. Consequently, the QC (Quality Control) can be carried out more frequently. Systems and software engineers rapidly learn, through SQC (Software/Specification QC) feedback, to take standards seriously, which in turn reduces defect injection. Further, by analyzing where/how the defects occur, continuous process improvement can be supported. The key idea is to inexpensively measure the degree of violation of critical practices, as expounded in standards ('Rules'). Then to make sure that work which exceeds reasonable levels of major defect density fails to exit from its creation process. Avoid Garbage Out!

Keywords: Continuous Process Improvement, Inspection, Peer Review, Sampling, Specification Quality Control.

1 Introduction

If we carry out inspection of specifications properly [1][2] the cost is barely tolerable for some: about one hour of effort, per page1 checked, per systems engineer or software engineer. The harvest, even if we are skilled, is only to identify between 40-80% of the major defects. That leaves many remaining major defects undetected, and many of these will be found, at considerable cost, during testing or in the final released product.

Of course, finding defects using traditional inspection (and fixing them) earlier than the test stage is beneficial and may even pay off. However, there is a better way: Specification Quality Control (SQC). It ought to appeal to all Spec QC purposes, and especially to the many organizations that have not been able to stomach the high costs and low effectiveness of traditional inspection.

The main concept of SQC is to shift emphasis from "finding and fixing defects" to "estimating the specification defect density", and using this information to motivate systems and software engineers to learn to avoid defect injection in the first place. Such a shift permits a dramatic cost saving. When our QC purpose is measurement rather than "cleanup" we can sample rather than have to check 100% of the specifications. This is the major opportunity that SQC provides. The main purpose of SQC is to motivate individuals to learn to reduce major defect insertion.

Secondary purposes include:

- To prevent uneconomic major-defect density specifications from escaping downstream – and thus to avoid the consequent delays and quality problems. The major tactic to achieve this is to impose a numeric exit-barrier for the specification process, such as "only a maximum of 1.0 remaining majors per page"
- To teach and reinforce current specification standards.

2 Process Details

2.1 Traditional Inspection Method

The old inspection method (widely practiced in CMM Level 3 as peer reviews) was based on the idea of inspecting 100% of all pages, at optimum rate checking (one page per hour), using a review team of between 2 and 5 software engineers.

1 A page is defined as 300 words of non-commentary text. Non-commentary text is core specification or background text; it is not notes or other commentary text.

Authors

Tom Gilb is the author of "Competitive Engineering: A Handbook for Systems & Software Engineering Management using Planguage" (2005), "Principles of Software Engineering Management" (1988) and "Software Inspection" (1993). His book "Software Metrics" (1976) coined the term and, was used as the basis for the Software Engineering Institute Capability Maturity Model Level Four (SEI CMM Level 4). His most recent interests are development of true software engineering and systems engineering methods; and teaching top managers better ways to control big projects (better objectives and Evolutionary learning). Tom Gilb was born in Pasadena CA in 1940. He moved to England in 1956, and then two years later he joined IBM in Norway. Since 1963, he has been an independent consultant and author. <Tom@Gilb.com>, <www.gilb.com>.

Lindsey Brodie is an IT consultant, currently carrying out PhD research at Middlesex University. She has worked in industry for many years: originally in technical support and then in software development, corporate IT strategy and business process design. She edited Tom Gilb’s 'Competitive Engineering' book. Her interests include requirements and stakeholder value. Her current research concerns prioritization using metrics; the aim being to improve support for stakeholder IT decision-making. <L.Brodie@mdx.ac.uk>.
and systems engineers. The maximum yield of major defects from such an inspection process is in the range of 40%–80% depending on specification type (for example, a maximum of 60% for software source code specifications, and a maximum of 80% for requirements specifications – in practice however it is actually more likely that only 30% is achieved since malpractice is common). The reported ability to actually correctly correct major defects once found is only 5 out of 6 fixes attempted ([3], reported in [1]). All this amounts to the following:

- The same order-of-magnitude defects remaining as before the quality control process was applied
- Little or no change in the defect insertion density. In requirements specifications this regularly exceeds 100 major defects per 300 lines of specification (Personal experience by field measurement over many years).

2.2 New SQC Method

The new "SQC method" is based on the following:

- Sampling of a specification.
- A few (1 to 3) pages at a time.
- Starting early (perhaps once the first 5% of a large specification is written).
- Frequently (every week or so) until the work is completed.

For each individual systems or software engineer (each one must be motivated and trained personally), their sampled specification pages will be checked against a set of a few simple rules – usually about 3 to 7 rules are applied (for example, for initial checks these could be: Clear enough to test, Unambiguous to intended readers, and No design options in the requirements). The reviewers/checkers are asked to identify all deviations from these rules. Any deviation is termed a "specification defect". The reviewers/checkers are then asked to classify any specification defect that can potentially lead to loss of time or significant reduction in product quality as "major". The entire checking session might use only 2 engineers for 30 to 60 minutes. This might seem quite a high checking rate, but remember that only a few rules are being used and no other documents are being consulted to check out the original source of material, so we can go faster. In any event, as long as we turn up more defects than the threshold exit level for defects, then exactly how effective we are in detecting defects is a secondary issue.

Major defect findings are reported to a review leader, who calculates the estimated number of defects actually present, based on the total found by the team. An inexperienced team is usually about one third effective, so the estimated total number of majors per page is about three times the total of unique majors found by the team. This is a very rough calculation but it seems to work well in practice.

A pre-arranged standard for exit control (the fail to exit level) is set for unacceptable specification major-defect density. Initially it can be set at "anything more than 10.0 majors per page". In the longer term (beyond 6 months of culture change), the aim should be to set the limit at "anything more than 1.0 majors per page". To give some examples, IBM reported using a maximum of 0.25 major defects per page [4]. NASA reported a standard of using 0.1 major defects per page [5]. The initial limit set is a matter of trying to get better as fast as humanly possible. Ultimately, it should become a matter of finding the level that pays off for the class of work you are doing.

Note: There are several limitations to this simplified SQC process:

- It is only a small sample, so the accuracy is not as good as for a 100% or for a larger-than-few-page sample
- The team will not have time or experience to get up to speed on the rules and the concept of major defects
- A small team of two people does not have the probable greater effectiveness of 3 or 4 people
- The entire specification will not have been checked, so there will not be the basis for making corrections to the entire specification
- The checking will not have been carried out against all the possible source documents (usually in the SQC process no source documents are used and memory is relied on. While this means that the checking is not nearly as accurate, it does considerably speed up the process).

However, if the sample turns up a defect-density estimation of 50 to 150 major defects per page (which is quite normal), that is more than sufficient to convince the people participating and their managers that they have a serious problem.

As discussed earlier, the immediate solution to the problem of high defect density is not to set about removing the defects from the document, because the same order of magnitude level of defects would still remain. The best solution for a document with a high defect density is to rewrite it entirely, using an individual who does not insert too many defects. Long term, the most effective practical solution is to adopt SQC as part of the corporate process, and more importantly, to make sure each individual specification writer takes the defect density criteria (and its "no exit" consequence) seriously. They will then learn to follow the rules, and as a result will reduce their personal defect injection rate. On average, a personal defect injection rate should fall by about 50% after each experience of using the SQC process. Widespread use of SQC will result in large numbers of systems and software engineers learning to follow the rules. To get to the next level of quality improvement, the next step is to improve the rules themselves (see Figure 1).

3 Case Study 1: A Financial Organization

In 2003, a large multinational financial group was a pilot user of this SQC process (see Figure 2). It also had combined this with adopting a specification and planning language, Panguage [6]. After six months, the organization reported the following for requirements and design specifications:

- Across 18 development projects using the new requirements method, the average major defect rate (per page)
SQC Process


Entry Conditions:
- A group of two, or more, suitable people* to carry out SQC is assembled in a meeting.
- The people have sufficient time to complete an SQC. Total Elapsed Time: 30 to 60 minutes.
- There is a trained SQC team leader at the meeting to manage the process.

Procedure

P1: Identify Checkers: Two people, maybe more, should be identified to carry out the checking.

P2: Select Rules: The group identifies about three rules to use for checking the specification. (My favorites are clarity ("clear enough to test"), unambiguous ("to the intended readership"), and completeness ("compared to sources"). For requirements, I also use "no optional design").

P3: Choose Sample(s): The group then selects sample(s) of about one "logical" page in length (300 non-commentary words). Choosing such a page at random can add credibility – so long as it is representative of the content that is subject to quality control. The group should decide whether all the checkers should use the same sample or whether different samples are more appropriate.

P4: Instruct Checkers: The SQC team leader briefly instructs the checkers about the rules, the checking time, and how to document any defects, and then determine if they are major defects (majors).

P5: Check Sample: The checkers use between 10 and 30 minutes to check their sample against the selected rules. Each checker should "mark up" their copy of the document as they check (underlining issues and classifying them as "major" or not). At the end of checking, each checker should count the number of "possible majors" (spec defects, rule violations) they have found in their page.

P6: Report Results: The checkers each report to the group their number of "possible majors." Each checker determines their number of majors and reports it.

P7: Analyze Results: The SQC team leader extrapolates from the findings the number of majors in a single page (about 6 times** the largest number of majors found by a single person, or alternatively 3 times the unique majors found by a 2 to 4 person team). This gives the major-defect density estimate. If using more than one sample, you should average the densities found by the group in different pages. The SQC team leader then multiplies the "average major defects per page density" by the "total number of pages" to get the "total number of major defects in the specification" (for dramatic effect!).

P8: Decide Action: If the number of majors per page found is a large one (ten majors or more), then there is little point in the group doing anything, except determining how they are going to get someone to write the specification "properly", meaning to acceptable exit level. There is no economic point in looking at the other pages to find "all the defects", or correcting the majors already found. There are simply too many majors not found.

P9: Suggest Cause: The team then chooses any major defect and thinks for a minute why it happened. Then the team agrees a short sentence, or better still a few words, to capture their verdict.

Exit Conditions
- Exit if less than 5 majors per page extrapolated total density, or if an action plan to "rewrite" the specification has been agreed.

Notes:
- * A suitable person is anyone, who can correctly interpret the rules and the concept of "major".
- ** Concerning the factor of multiplying by "6 ": We have found by experience ([1]: reported by Bernard) that the total unique defects found by a team is approximately twice that of the number found by the person, who finds the most defects in the team. We also find that inexperienced teams using SQC seem to have about one third effectiveness in identifying the major defects that are actually here. So 2 x 3 = 6 is the factor we use (Or 3 x the number of unique majors found by the entire team).
14 of the 18 development projects exited successfully on first pass SQC. The other 4 development projects failed to meet the exit criteria of 10 major defects per page, the projects’ specifications had to be improved, and were then re-inspected.

A sample of 6 development projects with requirements in the "old" specification format were tested against the following set of rules:

1. The requirement is uniquely identifiable
2. All stakeholders are identified
3. The content of the requirement is "clear and unambiguous"
4. A practical test can be applied to validate delivery of the requirement.

The average major defect rate (per page) in this sample was 80.4.

A few months later, as a result of the continuing overall success of the pilot testing, the client decided to spread SQC widely to all types of technical specification.

4 Case Study 2: A Jet Engines Manufacturer

At one of my clients, we sampled 2 pages of an 82-page requirements document: four managers checked page 81, and four other managers, who were directly involved with the requirement specifications projects, checked page 82. These pages were all "non-functional" requirements (such as, security). We agreed to check against the following simple set of requirement specification rules:

1. **Unambiguous** to intended Readership
2. **Clear** enough to test.
3. No **Design** specifications (= "how to- be good") mixed in.

Violation of any one of these rules constituted a specification "defect" and was classified either as "major" (likely to result in potential damage to effort or quality) or "minor" (no way they can harm us, even though they are defects).

We also agreed a specification exit level of "No more than one remaining major defect per page". They "agreed" (for demo purposes!) that any manager who signed off (approved) a requirements specification with more than 100 remaining major defects per page should be fired for incompetence. Later that day they themselves were, as we shall see, to provide clear numeric evidence that – they themselves should be "fired"!

The 8 managers were given 30 minutes to check their page. At the end they reported the following major defects found by themselves individually:

- Page 81 (three quarters of a page): 15, 15, 20, and 4 majors.
- Page 82 (a full page): 24, 15, 30 and 3 majors.

4.1 Estimating the Number of Major Defects Found by the Team

From the results of this input, we could estimate the number of unique major defects found by the team. First we had a hypothetical choice of either logging all the unique
major defects (Using old Inspection methods, logging would take 3 minutes for each defect resulting in a 3 hour job), or estimating the result approximately. Not surprisingly, the managers choose the quick estimation. To estimate the number of unique majors (that is, the number of majors that are not duplicated - so if the same defect is found by more than one checker it only counts as one defect); we can double the count of the largest number of majors found by one individual in a small (2-4 people) group. This is based on observations done at Cray Research ([1] pp. 299-301).

From personal experience, it works well. In this case, this means that the group working on page 82 had about (2 x 30) 60 majors per page found (±15 majors of course). The group working on page 81 had about 40 total unique major defects they could log if they so chose to log them in detail.

4.2 Estimating the Total Number of Defects per Page – Including Those NOT Found by the Team

But of course inexperienced checkers do not find 100% of the majors defects present - they find only about a third. Remember even experienced checkers carrying out source code inspections peak at source code bug-finding effectiveness of 60% ([1] reported by IBM MN), and most groups are not that good. Requirements and design checking tends to have an effectiveness ranging between 30% and 80% or more, depending on a wide range of process factors. These effectiveness factors include, speed of checking, available related project data, use of standards and checklists, and intelligibility of the specification being checked.

4.3 Can we Verify the Level of Checking Effectiveness in Practice?

If you want to prove these estimates, the proof is simple: carry out an inspection, and then remove the major defects you have identified. That should leave twice that number estimated remaining – the two thirds NOT found by checkers (In this example, 80 major defects for page 81, and 120 for page 82). This sounds incredible. How could people miss so many on a single page? The proof comes when you repeat the checking process, and predictably find one third of the remainder (one third of 80); and can prove they were there on the first checking pass. Skeptics turn into believers at this point. We have carried out this test on our courses for years, and it always prove the case.

4.4 So, How Many Major Defects Are There in Total on the Page?

In this case, the managers accepted my assertion – that the 60 majors on page 82 were an indication of about 180 majors in the page (and 150 majors on page 81, indicative of the same density as page 82). Now this indicates an average of (120 + 180)/2 = 150 majors per page. I asked the managers if they felt this was probably typical for the other ("functional") pages. They said they had no doubts that it did. If managers are skeptical, the solution is simple, take another sample at random. I can assure you that the result found for defect density will be essentially the same order of magnitude.

4.5 Then, How Can We Estimate the Total Number of Major Defects in the Specification?

Now this leads us to an estimation that we have about 150 (average per physical page) x 82 (total pages) = 12,300 Majors in total. I was initially quite shocked on calculating this number. But the managers were for some strange reason not as skeptical as I was. I did not know anything about the project beyond that the requirements were just handed to me 45 minutes earlier and that the managers were somehow responsible.

4.6 How Many Bugs Will Be Generated as a Result of these Specification Major Defects?

But let"s carry on with the calculations! Now another factor that has to be taken into consideration is that not all major defects in specifications lead directly to bugs. The problem being that we don"t know exactly which of the major specification defects will actually cause bugs to be inserted - that depends on the "sleepiness of the programmers on the day"! Two pieces of research I recall showed that 25% to 35% of majors actually turn into bugs. For example, to make this plausible, a random guess as to the correct interpretation of an ambiguity with 2 options would give a 50% chance of a bug and 50% not. I have found that a good rule of thumb, one which correlates well with observed reality, is that one third of the major defects will cause bugs in the system. So, for this example, that implies that about 4,100 (= 12,300/3) bugs will occur.

4.7 What Do these Major Defects Cost in Project Terms? How Do They Delay the Project?

One of my clients (Philips Defence, UK; see the case study in [1], page 315) studied about 1,000 major defects found in the specification inspection of a wide variety of systems engineering specifications. They determined that the median downstream cost of not finding the majors would have been 9.3 hours (range up to 80 hours). So I use 10 hours as a rough rounded approximation of the cost of a major if it occurs downstream (at test and field stages).

Well, in this case study, that implies 41,000 hours (10 x 4,100 defects that hit us) of effort lost in the project through faulty requirements. I was quite shocked at the implication of this quick estimate based on a small sample. But the managers were quite at home with it. They responded, "Don"t worry Tom, we believe you!"

Why? I asked. So they explained, "Because (and we know you did not have any inkling of this) we have 10 people on the project, each using about 2,000 hours per year, and the project is already 1 year late (a total of 20,000 hours) and we have at least one more year of correcting the problems before we can finish."

5 Case Study 3: An Air Traffic Control Project (in Sweden/Germany)
Experiences and Advances in Software Quality

**Specification Quality Control (SQC) Form - An Example Filled Out**

SQC Date: May 29, 200X. SQC Start Time: ______
SQC Leader: Tom.
Author: Tino. Other Checkers: Artur.

Total Physical Pages: 10.
Sample Reference within Specification: Page 3.
Sample Size (Non commentary words): approx. 300.

Rules used for Checking: Generic Rules, Test Plan Rules.
Planned Exit Level (Majors per page): _______ or less.
Checking Rate Planned (Non commentary pages per hour): 2.
(Note this rate should be less than 2 pages per hour)

Actual Checking Rate (Non commentary words per minute): __________
Number of Defects Identified by each Checker:
   Majors: 6, 8, 3. Total Majors Identified in Sample: 17.
   Minors: 10, 15, 30.
Estimated Unique Majors Found by Team: 16 ± 5.
(Note 2 x highest number of Majors found by an individual checker)
Estimated Average Majors per Page: ~16 x 3 = 48.
(A Page = 300 Non commentary words)
Majors in Relation to Exit Level: 48/1 (47 too many).
Estimated Total Majors in entire Specification: 48 x 10 = 480.

Suggested Causes (of defect level): Author not familiar with rules.
Actions suggested to mitigate Causes: Author studies rules. All authors given training in rules.
Person responsible for Action: Project Manager.
SQC End Time: 18:08. Total Time taken for SQC: ________

Version: August 15, 2004. Owner: Tom@Gilb.com

Figure 3: An Example of an SQC Form Filled Out.

Another client had a seriously delayed software component for an air traffic control simulator. The contract dictated about 80,000 pages of logic specifications. The supplier had written and approved about 40,000 pages of these. The next stage for the logic specifications was writing the software.

The divisional director, Ingvar, gave me the technical managers for a day to try to sort out the problem. These men had each personally signed off the 40,000 pages. We pulled 3 random pages from the 40,000 and I asked the managers to find logic errors in the specifications – errors in the sense that if coded the ATC system would be wrong. Within an hour of checking they found 19 major defects in the 3 sample pages. They agreed that these pages were representative of the others.

That evening, Ingvar took 30 minutes to check the 19 defects personally, while his managers and I waited in his office. He finally said, "If I let one of these defects get out to our customer, the CEO would fire me!"

Now the 19 defects found in the 3 pages represent an actual defect density of approximately three times that (that is, they probably did not find two thirds of the existing defects). So the managers had signed off about (20 x 40,000) 0.8 million bugs. And they had only done half the contracted logic specification. Well, the sample told us a great deal.

We started thinking that afternoon about what could have been done better. The conclusion was that we had a "factory" of analysts producing about 20 major defects per page of ATC logic specification. We also concluded that if we had taken such a sample earlier, say after the first dozen
Estimating Remaining Major Defect Density

Assumptions:
A logical page (page) is 300 non-commentary words.
Your SQC effectiveness is 33.3% and your SQC is a statistically stable process.
One sixth of your attempts to fix defects fail (One sixth is average failure to fix).
New defects are injected during your attempts to fix defects at 5%.
The uncertainty factor in the estimation of remaining defects is ± 30%.
Probable remaining major defects per page = "Probable unidentified majors" + "Bad fix majors" + "Majors injected"
Let \( E \) = Effectiveness expressed as a percentage (\%) = 33.3%

If 33 major defects per page have been found during SQC.
Probable unidentified majors =
Major defects total estimated 3 x Found Majors (33) = about 100 ±30

Bad fix majors = One sixth of fixed majors =
Of 30 attempted fixes, 5 major defects in each page are not fixed.
This is useful to recognize.
Even if you found all defects, 1/6 would remain after all were fixed.

Majors injected = 5% of majors attempted to be fixed = 1.5 major defects per page.
(this is not always calculated, since it is small, compared to the error margin)

Probable remaining major defects/page, after fixing what we found in a sample =
66 (not found) + 5 (not fixed) = roughly 71 remaining major defects per page.

Taking into account the uncertainty factor of ± 30% and rounding down to the nearest whole number gives 50 Remaining Major Defects per Page

(\text{Minimum} = 50, \text{Maximum} = 92 \text{ remaining major defects per page}).

Figure 4: An Example of Calculating the Remaining Major Defects per Page.

6 SQC Estimations and Calculations
At this point, it is worth summarizing the overall SQC process of estimating and calculating. See Figures 3 and 4, which show how to arrive at the defect level for a specification and how to calculate the number of remaining defects in a specification respectively. The calculations shown are for yet another case study.

7 Continuous Process Improvement
Notice how towards the end of Figure 3 there are two questions concerned with analyzing the origin of the defects (that is, "Suggested Causes" and "Actions suggested to mitigate Causes"). The aim of these questions is to identify problems in work practices that need correction. This approach is identical to Capability Maturity Model Level 5, and to the Defect Prevention Process (see discussion of Mays in [1]).

In the Raytheon Study [7][8], this process improvement effort reduced rework costs, within about 7 years, from approximately 27% of all development costs down to about 4%. Before that happened though, the individual discipline of software engineers actually following their existing (bad) processes, led to a reduction from 43% rework costs to the 27% cited above in a year. So there is a lot of short-term improvement available by getting people to follow even sim-
Experiences and Advances in Software Quality

ple standards.

Personal experience with SQC is that by merely motivating people to follow the simple rules of "clear/unambiguous/no design" in requirements we can reduce the number of major defects inserted into requirements by, in one case, an average of 80 majors/page to about an average of 11 majors/page within 6 months. Corporate engineering measurements (Douglas Aircraft 1988) and other examples indicate that the individual rate of reduction of defect insertion is about 50% per learning cycle. So, in about 7 cycles of writing specifications and measuring defects an individual gets to the exit level of less than one major per page.

8 Summary

SQC costs very little, but its effect on early control over injected defects is significant. It can drive defect injection down by one and then, with time, two orders of magnitude.

The key SQC concept compared to traditional Spec Inspection methods is to measure by sampling, and use the information to motivate people to "learn the rules" (that is, the standards and/or best practices), and reduce their defect injection.

Traditional Spec Inspection techniques are doomed to high costs and low effects because:

- They can only hope to find about half the problems (Given 40-80% is the very best in practice)
- They spend approximately 3-4 hours engineering effort per page of specification (for full effectiveness)

References

Experiences and Advances in Software Quality

The Software Process Improvement Hype Cycle

Miklós Biró

This paper provides a historical perspective on the state of the field of software process improvement (SPI). Just as process improvement itself, the development of our expectations regarding process improvement can be viewed following a staged model which is analogous to the popular Gartner Hype Cycle for innovation. The stages highlighted in this survey are characterized by the issues in their primary focus which are mostly not forgotten at all in later stages but rather further expanded and becoming more mature. The characteristics of the identified stages are: awareness of process capability weaknesses triggered by the software crisis and CMM, SPI and ISO9000 expectations, bridging the trough of disillusionment, enlightenment leading to further recognition of the importance of business goals, plateau of spreading to other disciplines and models, trough of doubts and new triggers, plateau of reconciliation and industrial adoption. The hype cycle view of historical development can contribute to the appreciation of the role of various approaches to software process improvement, as well as to the better comprehension of the way their combination can benefit the industry.

Figure 1: Hype Cycle Phases Applicable to any Innovation.

It is a characteristic of human nature in uncertainty-tolerant cultures [4], that innovation triggers significant interest followed by inflated expectations which are naturally not fully met. The resulting disillusionment may be followed by a deeper understanding and sober adoption of the innovation which will have matured by then. This is the idea of the popular concept of Hype Cycle coined in 1995 at the Gartner information technology research and advisory company based in the U.S.A. Referring to the complete analysis of the subject [5], the phases of this hype cycle (see Figure 1), originally applied to emerging technologies but applicable in fact to any innovation, are:

1. “Technology Trigger”
2. “Peak of Inflated Expectations”

Author

Miklós Biró is associate professor with habilitation at Corvinus University of Budapest with over three decades of software engineering, university teaching (including professorship in the USA), and management experience. He has a Ph.D. in mathematics (operations research) from the Loránd Eötvös University in Budapest, an Executive MBA degree from ESC Rouen, France, and a Master of Science in Management degree from Purdue University, USA. He is fluent in Hungarian, English, and French. He has initiated and managed the Hungarian participation and consulting activities in numerous European projects and organizations. He is the author, co-author of Hungarian and English language books and numerous international publications in the fields of Software Engineering Management, Decision and Negotiation Support Systems, Combinatorial Optimization, etc. He is member of the Editorial board of the journal Software Process Improvement and Practice published by John Wiley & Sons, member of the Hungarian Doctoral Council and of the Public Body of the Hungarian Academy of Science. He is founding president of the professional division for Software Quality Management of the John von Neumann Computer Society, the Hungarian national representative in Technical Committee 2 (TC-2) Software: Theory and practice of the International Federation for Information Processing (IFIP) where he is coordinator of the Manfred Paul Award. <miklos.biro@uni-corvinus.hu>.

Keywords: Agile Development, Bootstrap, Business, CMM, CMMI, Hype Cycle, ISO 9000, ISO/IEC 15504, Manifesto, Maturity, Process Capability, Quality, SPI, SPICE.
3. "Trough of Disillusionment"
4. "Slope of Enlightenment"
5. "Plateau of Productivity"

The software process improvement movement started with the CMM being a significant innovation. The hype cycle was triggered at the end of the 1980’s and went through phases which do not entirely follow the one promoted by Gartner. The reason for the difference is the support and acceptance of the model by the U.S. Department of Defense which helped the CMM avoid the full trough of disillusionment by supporting continuous innovation in the form of the Capability Maturity Model Integration (CMMI) for example. CMMI is both the result and the further catalyst of the spreading of process maturity models to disciplines other than software development. This plateau of spreading to other disciplines and models is followed by the trough of doubts and new triggers like agile software development with a new hype cycle starting with new expectations. Feelings are still high, however the light of reconciliation on a new plateau of industrial adoption is already visible.

This paper gives an overview of the progress of software process improvement along the phases of the altered hype cycle:
1. Awareness of process capability weaknesses triggered by the software crisis and CMM.
2. SPI and ISO9000 expectations.
3. Bridging the trough of disillusionment.
4. Enlightenment leading to further recognition of the importance of business goals.
5. Plateau of spreading to other disciplines and models.
6. Trough of doubts and new triggers.
7. Plateau of reconciliation and industrial adoption.

The particularity of this software process improvement hype cycle is exactly the bridging of the trough of disillusionment by the supported continuous innovation of the CMM which led to version 1.0 of the CMMI in the year 2000 followed by new high expectations, a plateau of spreading to other disciplines, another trough of doubts, and the current new plateau of reconciliation and industrial adoption.

While this phasing may generate controversy due to overlaps in the case of various approaches, as well as to the breadth of opinions reflected in the vast literature of the field, it allows the structuring of results which are impossible to fully list in this paper. The exact phrase "software process improvement" gave 245,000 results in Google and 12,400 results show that only 32% of all projects were succeeding (i.e. delivered on time, on budget, with required features and functions), "44% were challenged (i.e. late, over budget, and/or fewer than the required features and functions) and 24% failed (i.e. cancelled prior to completion or delivered and never used.)" [7].

As mentioned in the introduction, the process improvement movement, intended to overcome the software crisis, was initiated by the SW-CMM developed under the leadership of Watts Humphrey at the Software Engineering Institute (SEI) [Humphrey, Sweet 1987]. Its fundamental recognition was that the quality of the process determines the quality of the product. This slogan became more and more accepted in industrial production in general and in the software industry in particular [8].

The supported continuous innovation of the CMM professionally bridging the trough of disillusionment of the generic innovation hype cycle, naturally following the peak of SPI expectations discussed in the next section, is illustrated by the following fact:

"The SW-CMM was retired on December 31, 2005.
All SW-CMM appraisal results from CBA IPI and SCE appraisals will expire on December 31, 2007.
After December 31, 2007, all SW-CMM ratings will be considered invalid and should not be advertised." [9].

Nevertheless, despite the innovations introduced by the later discussed Capability Maturity Model Integration (CMMI) and ISO/IEC 15504, the original SW-CMM maturity levels depicted in Figure 2 continue to determine the cognitive schemes of professionals.

2. Awareness of Process Capability Weaknesses Triggered by the Software Crisis and CMM

There is no need today to highlight the significance of software in our everyday life from the provision and consumption of services to the management of business processes. It is a commonplace consequence that the timely and cost effective development of good quality software is crucial for both software consumers and software developer organizations which coexist in an increasing number of businesses. It was already estimated in the 1990's, that in Europe, 70% of software development was carried out by organisations whose core activity was not software [6].

On the other hand, software is a special product whose development requires technical and management skills which lie outside the culture and resources of most enterprises. Let’s see just a few characteristics of software which make it special among industrial products.

- Mass production of software does not require any special consideration at design time, contrary to most industrial products.
- Wear and tear of software is not due to the physical impact of the environment, but mostly to obsolescence.
- The testing of software is both practically and theoretically far more complex than that of other products.

These are fundamental reasons for the software crisis which led to the recognition of the significance of process capability and organizational maturity by the end of the 1980’s.

One of the most cited proofs of the software crisis is the Standish Group Chaos Report which has been regularly published since 1994. Even the recently published 2009 results show that only 32% of all projects were succeeding (i.e. delivered on time, on budget, with required features and functions), "44% were challenged (i.e. late, over budget, and/or fewer than the required features and functions) and 24% failed (i.e. cancelled prior to completion or delivered and never used.)" [7].

3. SPI and ISO9000 Expectations

An important milestone of the software process movement in Europe was the establishment of the Bootstrap Institute in 1994. The significance of this step is described
Experiences and Advances in Software Quality

among others in the publication [10] on the European Software Quality Network. The BOOTSTRAP methodology was developed in the framework of the EU ESPRIT programme and combined the following approaches: Software CMM, ISO 9001/9000-3, European Space Agency Software Engineering Standard PSS-05-0, and DoD-STD-2167A.


In addition to the ISO 9000-3 guidelines, there exist other types of international software quality standards whose relationship to the ISO 9000 series is enlightening. It was already described in the introduction, that the evaluation of the quality of a software product is one of the basic issues in information technology. A system of product quality criteria was summarized in the ISO/IEC 9126 standard which is evolving today into the ISO/IEC 25000 series. The first level of the criterion hierarchy contains the following six elements: functionality, reliability, usability, maintainability, portability and efficiency. The business decisions that are supported by this standard are as follows:

- Does the software requirements specification adequately reflect the user requirements?
- Does the developed software satisfy the user requirements?

Biro and Turchanyi [8] highlight however fundamental business decisions which are not supported by systems of product quality criteria and which justify the existence of the ISO 9000 series of standards as well as that of software process assessment and improvement approaches which go beyond ISO 9000. These decision problems are:

- The customer’s decision problem: Is the supplier able

![Figure 2: The Original SW-CMM Maturity Levels.](image-url)
to sustain the reliability of its production?
- The supplier’s decision problem: How can we improve the reliability of the production?

ISO 9000 certification is intended to support the customer’s decision by focusing on the process rather than on the product. Nevertheless, certification is a yes/no decision which provides little support for the supplier’s decision problem. It is precisely software process assessment and the corresponding improvement action plan which serve the fulfillment of the supplier’s need.

4 Bridging the Trough of Disillusionment

One of the major criticisms of ISO 9000:1994 was that its introduction became a burden with the overwhelming "ISO bureaucracy" which was only meant to control the production and was not ready to adapt to the permanent change of processes, technology and customer demands. This problem led to heavy disillusionment in ISO 9000 during the 1990’s which is keeping up today as well despite the new 2000 versions which are theoretically much more flexible.

The blame for the continuing criticism rests with the consultants and auditors as well, who are rarely open to new paperless approaches otherwise permitted by the standard which can even be combined with model based process improvement like CMMI and the emerging ISO/IEC 15504 international standard.

One of the experiences successfully combining model-based process improvement with the achievement of ISO 9000 certification is described in [12]. The main lessons derived from this experiment are the following:
- The approach of considering the improvement of the maturity level as the principal objective and the achievement of ISO 9001 certification as a side-effect is valid from the efficiency point of view.
- Even if ISO 9001 certification is not the principal objective of process improvement, it is worth capitalizing on its high recognition by allocating appropriate resources to its achievement.
- According to international experiences, there is usually a significant decline of attention towards the quality system after the ISO 9001 certificate is granted. The approach of considering certification as a side-effect of overall process improvement helps avoiding this trap.

Disillusionment regarding the CMM can be accurately detected in the literature. Goldenson and Herbsleb [13] write: "Still, we detect more than a little discouragement about the pace of process improvement. About a quarter of our respondents say that ‘nothing much has changed’ since the appraisal. Almost half say there ‘has been a lot of disillusionment over the lack of improvement.’ Over 40 percent say that process improvement has been overcome by events and crises and that other things have taken priority. Almost three-quarters tell us that process ‘improvement has often suffered due to time and resource limitations’; over three-quarters say that process improvement has taken longer than they expected; over two-thirds say that it has cost more than they expected."

The SEI conducts however systematic research to measurably identify success factors and to uncover myths, and also to publish these results. This fact is well illustrated by the following quotation from [14]: "In our survey, we were able to compare success rates of organizations of various sizes operating in different sectors in order to see if these factors played a major role in determining success. Most organizations in the survey were in the commercial (23), government contractor (19), or government (12) sectors, and our results show no systematic differences in the success rates among these sectors."

These measurable and well published research results, together with considerable work invested into the development of the also well publicized Capability Maturity Model Integration in the year 2000, contributed to bridging the trough of disillusionment of the CMM hype cycle.

There was however another earlier mentioned SPI initiative, Bootstrap, whose life-cycle ended in 2003. Bootstrap and actually the CMM itself were on the other hand precursors of the international standardization initiative called Software Process Improvement and Capability Determination (SPICE) which went through a long and turbulent trial phase during the 1990’s. SPICE survived the trough of disillusionment due to a complete rework including generalization to all processes not restricted to those related to software.

The finally published standard is ISO/IEC 15504 whose new parts are still appearing nowadays. According to the decision of ISO in May 2009, 15504 will be replaced with the 31001 series of standards.

5 Enlightenment Leading to further Recognition of the Importance of Business Goals

ISO bureaucracy has already been mentioned as a burden preventing the achievement of business goals including flexible adaptation to the continual change of processes, technology and customer demands. After signs of disillusionment, other SPI approaches were also rediscovered in the context of their potential contribution to business success. And this is actually one of the gateways to level 4 in the reworked and published ISO/IEC 15504 standard as well.

This issue is systematically discussed among others in the chapter on "The Software Process in the Context of Business Goals and Performance" of the book written by Biró and Tully [6] still quoted by business consulting experts [15], as well as the paper by Biró, and Messnarz [16] which analyse the ways in which software process improvement can provide leverage for a firm from the financial, operating, production, market and human behavioural perspectives.

A further issue, which becomes highly relevant to the rising globalization of business operations, is the consideration of the differences in cultural value systems when introducing new management processes. This issue is discussed in the context of SPI in [4] and [7].

6 Plateau of Spreading to other Disciplines and Models
As already discussed, the continuous innovation of the CMM resulted in the publication of the CMMI in the year 2000 which was an answer to the need best expressed by a quotation from the publicly available [18] document itself: 
"Since 1991, CMMs have been developed for myriad disciplines. Some of the most notable include models for systems engineering, software engineering, software acquisition, workforce management and development, and integrated product and process development (IPPD).

Although these models have proven useful to many organizations in different industries, the use of multiple models has been problematic."

The most recent model integrated into the CMMI Framework is CMMI® for Services 2009 [19] which "draws on concepts and practices from CMMI and other service-focused standards and models, including:

- Information Technology Infrastructure Library (ITIL)
- ISO/IEC 20000: Information Technology—Service Management
- Control Objects for Information and related Technology (CobiT)
- Information Technology Services Capability Maturity Model (ITSCMM)"

The interaction of CMMI is however mutual with these and other standards and models whose most recent versions themselves refer to the popular process capability and organizational maturity framework.

As already mentioned, one of the innovations of the published ISO/IEC 15504 international standard (Information Technology - Process assessment) is precisely its generalization to all processes not restricted any more to those related to software. This feature resulted in the development of a number of business domain-specific models like Automotive SPICE and SPICE 4 SPACE.

New models are being developed like Enterprise SPICE which "will integrate and harmonize existing standards as determined by the stakeholders to provide a single process reference model and process assessment model that addresses broad enterprise processes. It will provide an efficient and effective mechanism for assessing and improving processes deployed across an enterprise." [20].

It must be highlighted that while CMMI, ISO/IEC 15504, as well as ISO 9000 are apparently developed independently, they have a natural and intended mutual influence on each other. On the one hand, the "SEI continues to work with industry and government to align the CMMI Product Suite closely with ISO/IEC 15504", on the other hand it "is also working with the international standards community to influence the continued development of 15504 along with key process reference models such as those for ISO 9001, ISO/IEC 12207, and ISO/IEC 15288" [21].

7 Trough of Doubts and New Triggers

Doubts in the effectiveness of approaches, which were summarily characterized as "heavyweight", culminated in the Agile Manifesto whose essence is well expressed by the following paragraph:
"On February 11-13, 2001, at The Lodge at Snowbird ski resort in the Wasatch mountains of Utah, seventeen people met to talk, ski, relax, and try to find common ground and of course, to eat. What emerged was the Agile Software Development Manifesto. Representatives from Extreme Programming, SCRUM, DSDM, Adaptive Software Development, Crystal, Feature-Driven Development, Pragmatic Programming, and others sympathetic to the need for an alternative to documentation driven, heavyweight software development processes convened" [22].

The Agile Manifesto highlights the imminent sources of disillusionment by pointing out the higher value it attributes to:

- individuals and interactions over processes and tools,
- working software over comprehensive documentation,
- customer collaboration over contract negotiation,
- responding to change over following a plan.

And there is an important additional sentence which opens the way to the following phase of the hype cycle: "That is, while there is value in the items on the right, we value the items on the left more."

In fact, agile methods have roots dating far earlier than the manifesto. Two of the most prominent representatives of the direct roots are Barry Boehm with the spiral model for software development [23], and Tom Gilb who was the first to argue for very similar principles in his evolutionary method for software engineering management already in the 1970's, and who published the recognized book [24] before the wide appearance of the CMM.

A clear effect of the agile movement is that it triggered a new hype cycle where it may itself approach the trough of disillusionment which has even received a rotund name: "death of agile", which may on the other hand be no more than the result of malpractice in many cases.

8 Plateau of Reconciliation and Industrial Adoption

There are numerous signs of reconciliation between the "heavyweight" and agile worlds starting with the CMMI itself, which recognizes both the spiral and the evolutionary development lifecycles, through the mentioned additional sentence of the Agile Manifesto, to the message of the book entitled "Balancing Agility and Discipline: A Guide for the Perplexed." [25].

A recent technical note published at the SEI [26] further recognizes that: "CMMI and Agile can complement each other by creating synergies that benefit the organization using them. Agile methods provide software development how-to's that are missing from CMMI best practices that work well—especially with small, co-located project teams. CMMI provides the systems engineering practices that help enable an Agile approach on large projects. CMMI also provides the process management and support practices that help deploy, sustain, and continuously improve the deployment of an Agile approach in any organization."

As far as reconciliation with the ISO/IEC 15504 com-
munity is concerned, doors have always been open shown among others by the pertinent abstract of a most recent paper [27]: "In the last two decades several models for evaluating software process capability have been defined and became more and more popular. The application of such models, and in particular the ISO/IEC 15504, determined a general software process improvement in many domains. Nevertheless, the application of the ISO/IEC 15504 standard is still considered by many agile developers as incompatible with agile approaches. Such an attitude is mainly based on common misunderstandings on what the ISO/IEC 15504 is and on what its application involves. This paper aims at showing that this standard, if genuinely applied, can be effectively used also in agile contexts."

Integrating all approaches with the aim of benefiting the software, systems and services industry, an associated event will take place at the annual European Systems & Software Process Improvement and Innovation Conference 2009 <http://2009.eurospi.net/> in Alcalá de Henares near Madrid, creating the Manifesto for SPI with initial supporters from a wide international community.

9 Conclusion

The last couple of decades were rather turbulent in the field of software process improvement among others. The hype cycle model can be used to cognitively domesticate the understanding of this turbulence as shown in this paper.

The future will hopefully lead to mutual appreciation of the special advantages of the different approaches to software process improvement as well as to better comprehension of the way their combination can benefit the industry.

References

[19] Software Engineering Institute. CMMI® for Services,
Experiences and Advances in Software Quality


Quality Going for Gold

Derek Irving and Margaret Ross

This paper will discuss the proposed changes in the TickIT scheme - TickITplus. These changes are currently out for consultation, following development over a period of several years. The reason for reconsidering the TickIT scheme stems from the need to be able to identify and differentiate between companies, as the level of quality process needs to improve both for clients and suppliers. The concept of higher levels of quality process has been established for many years with CMM and now with the CMMI models. The paper will consider the proposed four levels of TickITplus, Bronze, Silver, Gold and Platinum. The current TickIT scheme is similar in principle to the Bronze level so that current TickIT auditors will be able to perform audits at this grade after some initial retraining. Further training will be necessary to develop expertise in the TickITplus model and in order to audit at the higher assessment levels. Currently the documentation for the TickIT scheme comprises guidance material together with the criteria for the TickIT auditors. The TickIT Guide has been updated and maintained by the JTISC as TickIT has evolved, but this committee will be expanding the documentation to cover both guidance and requirements for the TickITplus scheme. Extension of the TickITplus scheme could provide opportunities for extending the scope of an audit visit to include other standards such as service management and IT security. This could minimize the cost and disruption for clients that are being audited against the various standards.

**Keywords:** Audits, Software Quality, TickIT, TickITplus.

1 Introduction

TickIT was developed in the early 1990s to provide quality assurance guidance in the software development field. It was a sector scheme based on BS 5750, which was later superseded by ISO 9001, which used trained auditors with software development experience. The TickIT Guide, which was frequently updated, gave guidance to IT systems developers, suppliers, purchasers and auditors. Although the scheme is still functioning, with many TickIT certificated organizations in the UK and elsewhere, the need for an update has been recognized.

Many software development companies have TickIT certification and the simple "pass/fail" approach of the underlying ISO 9001:2000 standard does not give potential clients any way of differentiating between them. Some major clients have been considering their own version of capability model, with second party audits, which is expensive for them and for their suppliers.

It is becoming increasingly common for joint audits to be carried out for related standards in order to save both time and money.

A project was therefore initiated in late 2007 which both updates and extends the original scheme. This is TickITplus,[1] which is being developed by the Joint Industry TickIT Steering Committee (JTISC), which is sponsored by the BCS, BSI and Intellect. The authors are members of this committee.

The content of the paper has been derived largely from TickITplus, A New Approach to IT Certification, issue 2. This was initially developed to support a series of seminars to explain scheme to auditors, practitioners and other interested parties.

Authors

Derek Irving is the Project Manager and principal architect of the TickITplus scheme. Trained as an engineer in the Royal Navy, he has served on both surface ships and nuclear submarines. A registered TickIT Auditor since 1993, he has assessed all variety of large and small organizations across the world. Now principally retired and once the TickITplus scheme is established, Derek plans to pursue a new career in art. He is a Chartered Engineer and Chartered Quality Professional, member of the IET and the CQI and holds a BA in Mathematics. <dkirving@iee.org>.

Margaret Ross is Professor of Software Quality at Southampton Solent University. Margaret’s original degrees were in Mathematics. Margaret’s area of interest is quality within a computing context. Since 1992 she has been Conference Director of the annual series of Software Quality Management international conferences, aimed at benefits to industry, and since 1995 of the annual series of international educational INSPIRE conferences. She has edited over thirty books. In addition she is a longstanding independent member of the Parliamentary IT Committee. Margaret has been and is influential in the British Computer Society (BCS), currently holding various positions including that of nationally elected member of the BCS Council, Vice Chair and Secretary of the BCS national Quality Special Interest Group, and member of the TickIT Committee. She was awarded an Honorary Degree of Doctor from Staffordshire University in 2004, an Honorary FBCS by the British Computer Society in 2007, and an MBE in the Queen’s Birthday Honours list in 2008. <Margaret.Ross@solent.ac.uk>.

2. General Concepts of the Scheme

IT activities in organizations have been expanded and diverged for over fifteen years. The development of software is no longer the only area of concern. Other activities and disciplines such as service management, information
security and IT infrastructure are also now potentially critical to organizations. The CMMI approach to capability assessments has become well established as a more sophisticated approach to determining the maturity of the processes of organizations. Although it is generally well regarded, it is still very expensive and does not come within the surveillance and regulation infrastructure of ISO 9001 certification.

TickIT had a considerable impact in the early 1990s on the software engineering community in the UK and it was made mandatory by several major purchasers. At that time, TickIT certification was treated as a requirement where software development was identified as part of the scope statement, but the relaxation of this rule, the widening scope of IT, and a more general move away from the benefits of a straightforward ISO 9000 certification, have had the effect of diminishing the importance of TickIT to many users. With this loss of confidence, comes the inevitable pressure on costs and if the benefits were no longer there, ISO 9001 and TickIT certification might not be justified.

There are many differences between TickIT and TickITplus. It is not just a minor adjustment of the original TickIT scheme but a whole new approach to the issue of enabling, measuring and certifying quality assurance in the IT arena: for developers, suppliers of IT products and services, for their customers, auditors and for the suppliers of certification services.

The project to develop TickITplus is run by the Joint TickIT Industry Steering Committee (JTISC), which is made up of a number of key stakeholders from all aspects of IT activity. It is sponsored by the BCS, Intellect and the BSI, and in addition the project has had UK Government funding.

2.1 Process Capability

A major change is the introduction of process capability assessment into certification. Instead of the CMMI approach, ISO/IEC 15504: Process Assessment, is being used as the method of determining process capability and organizational maturity.

This is a five level approach which has been adapted for TickITplus to provide four grades of assessment: Bronze, Silver, Gold and Platinum (see Table 1).

<table>
<thead>
<tr>
<th>ISO 15504 Capability Level</th>
<th>TickITplus Capability Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1: Performed process</td>
<td>Bronze</td>
</tr>
<tr>
<td>Level 2: Managed process</td>
<td>Silver</td>
</tr>
<tr>
<td>Level 3: Established process</td>
<td>Gold</td>
</tr>
<tr>
<td>Level 4: Predictable process</td>
<td>Platinum</td>
</tr>
<tr>
<td>Level 5: Optimizing process</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Capability Levels according to TickITplus.
There is also a shortage of good practice information backed up by available quality material. By making a large part of the scheme documentation available for download free or at low cost, it is intended to increase the appeal and popularity of the scheme. Added to this, the scheme will encourage organizations to develop and self assess their processes, utilizing good practice in a defined framework. Assessments by approved but non-accredited assessors will also be an option. Having adopted either or a combination of these assessment approaches, the formal scheme would make provision for a fast track approach to full certification.

2.5 Requirements Based Scheme
At present, TickIT is known as a "sector scheme". This means it adds on to basic ISO 9001 certification an additional structure of qualified auditors, a training infrastructure, and guidance material. Nothing about TickIT, other than the need to use registered TickIT Auditors, is mandatory. This has always been seen as a limitation which restricts the benefits of the scheme. TickITplus brings mandatory requirements across the scheme's activities and grades. An organization can only reach and sustain a given grade if they have been formally audited as having fully achieved the requirements for that grade. The assessments will be based on quantifiable evidence of performance.

2.6 Scheme Regulation
There is a greater degree of central regulation planned for TickITplus. BSI and the International Register of Certified Auditors (IRCA) [5] are still planned to be involved in aspects of document production and auditor registration, but the scheme itself will remain under the overall control of the JTISC committee. Auditor registration will be treated as a specialist qualification, additional to ISO 9001, with external training providers and independent examinations. Certification bodies will work under UK Accreditation Service (UKAS) [6] accreditation to deliver the defined scheme across different grades and requirements. The documentation for the scheme, its rules and regulations governing how all those involved will contribute to its function, will be under the overall control of JTISC.

2.7 Revised Qualifications and Types of Registration
Part of the focus of TickITplus is to improve both auditor performance and status. Grades aligned with the four performance grades will be introduced and a new category of "Practitioner" will be introduced; this either for internal staff such as quality managers or for independent consultants. In line with the need to extend the scope of TickITplus into a much broader remit, the structure of qualifications under the SFIA model [7] will be introduced in a number of "Skills Profile" areas by which auditors and practitioners will be categorized, SFIA (Skills Framework for the Information Age) is an industry structure model, developed by the UK Government, the BCS and other bodies, which covers all types of IT related occupations and lists the knowledge and skills required at level). These skills profiles will be used to define the allocation of auditors to assessments. The selected skills profile areas will be:
- Legal and compliance,
- Service management,
- Systems and software development,
- Project and programme management,
- Corporate strategy planning and management,
- Information management and security,
- IT systems engineering and infrastructure.

In addition there will be a core body of knowledge covering quality, contractual issues, procurement and general management areas. As the auditors reach higher grades, they will be expected to increase both their general knowledge and attain specialist skills and qualifications.

2.8 Training Provision
The training provision will be focused around the key roles of the Auditor and Practitioner and more be offered at different levels as related to the appropriate assessment grading required. Registration of training providers will be provided and monitored by JTISC. It is expected that there will also be specialist training needed, such as for ISO/IEC 20000 or other standards. The focus of the TickITplus training plans will be on requirements of the scheme. Examinations will be separately controlled and delivered independently of the training providers.

To ensure that the latest information is available, there will be a TickITplus website with the current registration details, support Information, and downloadable material. It will also include information about a proposed user club which will provide regular forums and a vehicle for the dissemination of knowledge and exchange of experience between members.

2.9 Process Engineering Support
Although a feature of ISO 9001, the area of process definition and management will be new for many people. In order to support users and to provide a consistent content and format for the assessment, the scheme will provide a core Base Processes Library (BPL). This will be based around the basic definitions and structure in ISO/IEC 12207 and ISO/IEC 15288, in addition to other processes as needed. This will define the processes and their elements that are required to generate a compliant Process Reference Model (PRM), which is a key requirement of the scheme which all users will need to develop. There will also be key references to all the included Requirements Standards covered. This will define the mandatory process areas needed to comply with the selected scope and assessment grade. In order to aid the building of Process Model and to insure consistency, Scope Profiles, following the same format of the Skill Profiles described previously, will be used as the basic building blocks of the scope definition. The Base Processes Library will also be used in the generation of a Process Assessment Model (PAM), used by the auditors. The Base Processes Library will be maintained...
by TickITplus and its use in the scheme will be mandatory.

2.10 Process Capability Assessment

Process assessment is based on attributes as defined by ISO/IEC 15504-2. These vary as grades increase. In order to conform to a specific TickITplus grade, capabilities in a range of relevant scope-determined processes need to be demonstrated. The weighting given to these processes is determined by the type, which may be one of the following:

Type A, which is mandatory, mandated by ISO 9001.
Type B, which is mandated by selected scope and Requirements Standards.
Type C, which is dependent on the supporting scope.
Type M, which is maturity measurement, mandated by ISO/IEC 15504 at Gold and Platinum level.

The Base Processes Library provides the information on which processes and which of their attributes are required at the particular level and scope. Provision within the assessment methods will provide for the limitations on cost and requirements of small organizations, especially at the Bronze grade. Assessment and cost will increase for the higher grades.

2.11 Grade Progression

While adopting the grade approach defined in ISO/IEC 15504 provides organization maturity, TickITplus provides a clear progression route for both small and large organizations.

The first stage for many organizations will be a self-assessment. This will involve the ability for the organizations to use the tools and scheme techniques while retaining all normal in-house activities and working at the organization’s own pace. The next stage would be independent assessment, short of full certification although using registered assessors and having a formal recognition of achievement. This grade may also be used as a pre-assessment for later full certification.

The Bronze assessment is fully accredited certification to ISO 9001 and uses formal assessment methods from ISO/IEC 15504 together with a reduced sampling approach. All the mechanisms of TickITplus will apply, but with a reduced level of process assessment. This will be the normal entry grade to the TickITplus scheme. This is further extended under the transition arrangements operating for the existing TickIT certified organizations.

Assessments from Silver to Platinum are fully accredited certification to ISO 9001 and fully compliant with ISO/IEC 15504 at levels 3, 4 and 5.

Various fast-track approaches will be available for those organizations wishing to achieve a graded full certification and have met certain criteria at the self or independently assessed levels.

2.12 Migrating to TickITplus

The grades for TickITplus have been aligned to make the transfer from TickIT as straightforward as possible. Once the scheme has been accredited at Bronze level, there will be a nominal one year period for existing TickIT certified organizations to transfer to the new scheme, subject to a degree of discretion by their Certification Body. This will initially be an optional ungraded TickITplus. This will offer accredited certification and meet basic process requirements and a further three-year assessment period to fulfill the requirements for TickITplus Bronze grade. Hence existing TickIT certified organizations will have a minimum of four years to fully comply with requirements if they are to remain in the scheme.

New entrant organizations, or those returning after a break in certification, will need to meet the full Bronze or higher level requirements at the initial audit. However achievement at Platinum level requires an assessment and improvement history under full TickITplus. This will not be eligible for a direct entry.

While progression upgrades will be encouraged, they will be optional. Organizations at the Bronze level will be able to remain at that grade. Over time Silver is expected to become the predominant grade with relatively few organizations reaching the Platinum level. Those attaining Gold level are expected to be interested in progressing in time to Platinum.

All existing TickIT auditors will be eligible to transfer to TickITplus Bronze grade after taking the initial training course and satisfying the examination requirements. This needs to be done within three years following scheme accreditation, after which the existing TickIT auditor registration will cease. There will be no compulsion for auditors to pass beyond the Bronze grade.

The target launch date for TickITplus as an initial, non-accredited scheme is by 2010. At this time there will be auditors able to assess at the Bronze and Silver grades. Formal Foundation and Intermediate training courses will be available soon after that, together with the necessary examinations. Sufficient scheme documentation will be in place and the website established. It is then planned for formal accreditation to follow within six months at which point the termination of the existing TickIT certification will commence. Further information about the scheme and its progress will be available constantly at the TickITplus website.

3 Future Green Issues

There is a greater awareness of greening issues, particularly relating to computing. There has been considerable publicity indicating that the carbon footprint of the use of IT by industry is similar to that generated by the aircraft industry. There is evidence from various surveys that senior management have raised green issues high on their major concerns at Board level, particularly for large organizations. The benefits of addressing greening issues have been clearly identified, relating to the reputation of the organization, as well as preparation for potential legislation and to minimize cost. There is an awareness that data centres of large organizations have been responsible for the high cost of power usage and the waste of generated heat. Environ-
mental standards such as ISO 14000 [8] and five level models [9] could in the future form part of a future enhancement of the TickITplus scheme.

Small businesses could benefit by considering their green credentials, not only as potential financial savings but also to improve their reputation and credibility with their employees, the wider public, and their current and potential customers.

This move towards green computing for SMEs could be motivated by the potential pressure from larger organizations, who wish to be able to confirm that their suppliers and sub-contractors are also adopting green computing practices. This process has led to smaller organizations adopting quality standards, such as TickIT, in many cases under direct pressure from large organizations who refused to deal with companies that could not prove that their computing practices followed the appropriate quality standard.

References
Can Teamwork Management Help in Software Quality and Process Improvement?

Esperança Amengual-Alcover and Antònia Mas-Pichaco

In modern organizations teamwork is considered a key factor for success in business. A growing interest on team culture has led to a great number of contributions where different teamwork aspects are analyzed as drivers for teamwork practices improvement. Software development process is a team activity. Consequently, success in software organizations depends largely on the performance of software teams. In this article, firstly we study the teamwork key factors for success and quality in software development projects. Secondly, we present a teamwork assessment model for software teams.

Keywords: Software Process Improvement, Software Quality, Teamwork.

1 The Importance of Teamwork in Software Development Projects

Software engineering has long been recognized as a human activity which is managed through a system of processes and tools. The interaction and dependencies between the processes, the technology used to support them and the people implementing these processes represents the socio-technical environment of a software development project. The integration of the three elements of the "people, process and technology triangle" is usually considered by managers the basis for successful IT projects (see Figure 1).

In software companies technology is usually considered crucial to the effectiveness of their processes and, consequently, technology is normally used to support software development processes (e.g., CASE tools). However, the use of tools can only produce significant gains in software development projects if they are used in an appropriate manner which usually means the implementation of a strategy, a procedure or a well-defined process. Therefore, it seems that we give more importance to the second critical element, the process. In fact, process orientation is the current tendency in Software Engineering literature. Different process improvement models have emerged in recent years and they are becoming, step by step, a reference framework for software development organizations.

In this article we will focus our attention on the third critical element of the triangle: the people. Only people are able to make good use of the other two elements in a balanced way. In our particular triangle, the people dimension is the basis, the most important element (see Figure 2).

Multi-disciplinary research explores how cooperative and human aspects affect software development. A reasonable number of works consider human aspects a key factor in software engineering projects. In [1], its authors highlight that "it appears the human aspects of software development are more important that the technological aspects for better performance". According to [2] the majority of problems in software projects "are due to people problems, not technical ones". Consequently, bearing in mind that IT projects are usually a team activity, the good performance of teams should be considered essential for the success in these kind of projects.

2 Teamwork Key Factors

A large number of research projects address fundamental issues about software teamwork. Different authors expose in their studies the elements that should be taken into consideration to efficiently work in a team.

Larson and Lafasto’s research on high-performance teams [3] determines the eight dimensions of an effectively
functioning team. Those eight dimensions include:
1. A clear and elevating goal.
2. Results-driven structure.
3. Competent team members.
4. Collaborative climate.
5. Unified commitment.
7. External support and recognition.
8. Principled leadership.

Other relevant contributions to the analysis of team performance in organizations have been Belbin’s and Constantine’s role theories identifying the different roles in a team [4][5].

Steve McConnell, in his book *Rapid Development* [6], specifies the characteristics of a hyperproductive team:
- Shared, elevating vision or goal.
- Sense of team identity.
- Results-driven structure.
- Competent team members.
- Commitment to the team.
- Mutual trust.
- Interdependence among team members.
- Effective communication, a sense of autonomy.
- Sense of empowerment.
- Small team size.
- High level of enjoyment.

Lakhanpal has proved that team cohesiveness contributes more to productivity than the capabilities or the experience of the project members [7].

According to Barry Boehm, motivation has a larger impact on productivity and quality than any other factor [8]. Despite the number of studies emphasizing the importance of teamwork in software projects, it appears that there is not a consensus among authors to identify the characteristics that define teamwork.

With the goal of assessing teamwork, it has been necessary to precisely define the set of teamwork factors to be considered in order to measure and improve teamwork. As a result of the revision of the existing literature we have identified the following four factors as the "teamwork key factors" (see Figure 3):
- Management.
- Composition.
- Communication.
- Motivation.

3 Teamwork in Process Maturity Models

Software process improvement initiatives based on international standards for process assessment and improvement, such as CMM (Capability Maturity Model) [9] or SPICE (Software Process Improvement Capability dEtermination) [10][11][12][13][14], are focused on the assessment and improvement of the software lifecycle processes and do not explicitly consider essential aspects of teamwork.

After developing the Capability Maturity Model as a descriptive model of the characteristics of an organization at a particular level of software process maturity [15], the Software Engineering Institute (SEI) has developed the Team Software Process℠ (TSP℠), a prescriptive model for software development teams. As it is defined in the SEI technical report which relates the TSP to the CMM [16], "TSP is a high-maturity process for project teams. It contains an adaptable set of processes, procedures, guidelines, and tools for project teams to use in the production of high-quality software on time and on budget". In [17] some results from projects that have adopted the TSP are provided. The results show that TSP teams are delivering essentially defect-free software on schedule, while improving productivity.

Regarding applying the other standard, SPICE [18][19], our particular experience in leading software process improvement initiatives has brought us to consider teamwork an important aspect in any process improvement initiative.

4. Teamwork Assessment in Software Projects

With the interest of focusing on teamwork aspects in...
our future software process improvement initiatives, we have developed a teamwork assessment model for software projects. This model comprises a reference model, a questionnaire and a measurement scale.

4.1 A Teamwork Reference Model
The teamwork reference model presented in this article details the factors which should be considered in order to assess teamwork. Each factor of this reference model provides information in the form of:

a) a factor identification;
b) a factor name;
c) a description which details the different aspects for the factor; and
d) a set of teamwork best practices which identify the tasks needed to accomplish the purpose of the factor.

The following four tables show the detail for each one of the four teamwork key factors (see Table 1 to 4).

4.2 A Teamwork Assessment Questionnaire
At a conceptual level, our intention is to measure teamwork in software projects from two different perspectives: the project manager and the team members. This measurement goal has been refined, moving from a conceptual level to an operational level, by posing questions which compose a teamwork assessment questionnaire. Each question addresses a particular aspect of a teamwork factor.

The questionnaire comprises 55 questions distributed in two groups:

- A group of questions for the manager of the team. The main goal of this set of questions is to collect the manager’s view about the composition, management and performance of the team.
- A group of questions for all the members of the team. The purpose of this second group of questions is to collect the different responses from all the members of the team. These responses address the four "teamwork key factors".

At a quantitative level, we have defined a measurement framework for the assessment of the teamwork aspects. Within this framework we have established four possible responses to each question: never, sometimes, often and very often.

Finally, the performance of each teamwork factor is measured using the rating scale proposed by SPICE where four ordinal points are understood in terms of a percentage (see Table 5).

5. Conclusions and further Work
It is essential that Software Engineering considers the importance of the people dimension in software projects. The best technological solution together with a good process definition does not guarantee the success of a project if this project is not implemented by a team which performs the process in an efficient manner. Therefore, teamwork measurement should be additionally considered in any software process improvement initiative.

After analyzing the state of the art in teamwork in software projects, in this paper we have identified the four "teamwork key factors". These factors compose a teamwork reference model that can be used as a framework for teamwork assessment and improvement.

Our future work is to analyze the usefulness and applicability of this teamwork assessment model. We are currently working in a new SPICE-based software process improvement programme in small and medium companies of the Balearic Islands. The teamwork assessment model is expected to be applied in parallel with process assessment.
<table>
<thead>
<tr>
<th>Factor ID</th>
<th>MAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Factor Name</td>
<td>MANAGEMENT</td>
</tr>
</tbody>
</table>
| Factor Description | • Definition of a common vision that provides the team an identity as a team.  
• Actions whose goal is to identify the activities and tasks of the project, as well as to define dependencies among them.  
  **Note 1:** Each member of the team needs to know the objectives that the other members expect of him.  
  **Note 2:** All the members must understand their roles and responsibilities and have to agree on how to perform their activities.  
• Actions for planning the resources to be used during the project development.  
• Establishment of a monitoring system to control the progress of the project and to assess the performance of the team so that all the members are conscious of their results and see the progress to their objectives in accordance with the plan. |
| Best Practices | **MAN.BP1:** Evaluate feasibility of the project.  
**MAN.BP2:** Define the scope of work.  
**MAN.BP3:** Define the project schedule.  
**MAN.BP4:** Establish the project plan.  
**MAN.BP5:** Implement the project plan.  
**MAN.BP6:** Establish organizational commitment for measurement.  
**MAN.BP7:** Determine and maintain estimates for project attributes.  
**MAN.BP8:** Identify and monitor project interfaces.  
**MAN.BP9:** Monitor project attributes.  
**MAN.BP10:** Review the progress of the project.  
**MAN.BP11:** Act to correct deviations.  
**MAN.BP12:** Perform project close-out review.  
**MAN.BP13:** Evaluate and communicate information products and measurement activities to process owners.  
**MAN.BP14:** Evaluate staff performance.  
**MAN.BP15:** Conduct joint reviews. |

**Table 1:** The Management Factor.
### Factor ID CMP

<table>
<thead>
<tr>
<th>Factor Name</th>
<th>COMPOSTITION</th>
</tr>
</thead>
</table>

#### Factor Description

- Identification and definition of the different roles that can be assigned to the different members of the team.
  
  **Note 1:** Define the technical, management and collaboration skills necessary to perform each role.

- Selection of the most suitable and competent person for each team role.

- Assignation of responsibilities and authorities to the different members defining the team hierarchy.

  **Note 2:** Each member needs to understand the tasks and responsibilities of his/her role as a team member.

- Training aspects to assure that the members of the team have the knowledge and the necessary skills to perform their tasks in the team.

#### Best Practices

- **CMP.BP1:** Define needs for experience, knowledge and skills.
- **CMP.BP2:** Allocate responsibilities.
- **CMP.BP3:** Identify needed skills and competencies.
- **CMP.BP4:** Define evaluation criteria.
- **CMP.BP5:** Develop staff skills and competencies.
- **CMP.BP6:** Define team organization for projects and tasks.
- **CMP.BP7:** Maintain staff records.
- **CMP.BP8:** Develop a strategy for training.
- **CMP.BP9:** Identify needs for training.
- **CMP.BP10:** Develop or acquire training.
- **CMP.BP11:** Prepare for training execution.
- **CMP.BP12:** Train personnel.
- **CMP.BP13:** Maintain staff training records.

**Table 2:** The Composition Factor.

---

To analyse if teamwork improvement can help in process improvement.

**Acknowledgements**

This work was supported by TSI-030200-2008-4 "Plan para la evaluación, mejora de procesos y certificación según la norma ISO/IEC 15504", FIT-340502-2007-10 "acción asociativa estratégica de orientación a la excelencia tecnológica en el desarrollo software mediante la implantación de modelos de certificación de calidad (CMMI, SPICE)", and CICYT TIN2007-67843- TIN2007-67843-C06-04 "Modelos de simulación basados en ontologías y mejora de procesos para arquitecturas orientadas a servicios".
<table>
<thead>
<tr>
<th>Factor ID</th>
<th>COM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Factor Name</strong></td>
<td>COMMUNICATION</td>
</tr>
</tbody>
</table>
| **Factor Description** | • Identification of suitable communication mechanisms (interfaces, tools, software to transfer information, progress reports, joint reviews, etc.).  
**Note 1:** The members of the team need to be continually well informed about the activities of their colleagues in the team. They also have to have the opportunity of express their interests, worries and suggestions.  
• Reporting decisions.  
• Communicating changes to all affected parties. |
| **Best Practices** |  
COM.BP1: Communicate software requirements.  
COM.BP2: Communicate system requirements.  
COM.BP3: Communicate system architecture design.  
COM.BP4: Confirm system readiness.  
COM.BP5: Communicate modifications.  
COM.BP6: Collect feedback.  
COM.BP7: Establish organizational commitment for measurement.  
COM.BP8: Communicate measurement results.  
COM.BP9: Evaluate and communicate information products and measurement activities to process owners.  
COM.BP10: Provide feedback on performance.  
COM.BP11: Disseminate knowledge assets.  
COM.BP12: Establish the asset storage and retrieval mechanisms.  
COM.BP13: Notify re-users of asset status.  
COM.BP14: Distribute the results.  
COM.BP15: Track actions for review results.  
COM.BP16: Distribute documents. |

**Table 3:** The Communication Factor.

**References**


### Table 4: The Motivation Factor.

<table>
<thead>
<tr>
<th>Factor ID</th>
<th>MOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Factor Name</td>
<td>MOTIVATION</td>
</tr>
</tbody>
</table>
| Factor Description | • Actions addressed to excite the members of the team with the objective of getting their commitment to the project.  
• Opportunities for the development of the skills and competencies of the members of the team.  
• Empower all the members of the team to perform their job and develop their creativity. Promote participation.  
• Public recognition of the team’s efforts. |
| Best Practices | MOT.BP1: Ensure sharing of common vision.  
MOT.BP2: Enable active participation.  
MOT.BP3: Develop staff skills and competencies.  
MOT.BP4: Empower project teams. |

---


<table>
<thead>
<tr>
<th>Value</th>
<th>% achievement</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Not achieved</td>
</tr>
<tr>
<td>P</td>
<td>Partially achieved</td>
</tr>
<tr>
<td>L</td>
<td>Largely achieved</td>
</tr>
<tr>
<td>F</td>
<td>Fully achieved</td>
</tr>
</tbody>
</table>

Table 5: Teamwork Factors Measurement Scale.
Experiences and Advances in Software Quality


Evidence-based Software Engineering and Systematic Literature Reviews

Barbara Kitchenham, David Budgen, and O. Pearl Brereton

In 2004-5, Kitchenham, Dybå and Jørgensen wrote three papers discussing the concept of evidence-based software engineering (EBSE). EBSE is concerned with the aggregation of empirical evidence and uses systematic literature reviews (SLRs) as a methodology for performing unbiased aggregation of empirical results. This paper presents the concepts of EBSE and SLRs. In order to access the current impact of these concepts we relate existing systematic reviews to the software engineer’s body of knowledge (SWEBOK) structure. Our long term goal is to see the SWEBOK supported by a software engineer’s body of evidence.

Keywords: Evidence-based Practice, Evidence-based Software Engineering, Systematic Literature Review.

1 Introduction

In 2004, Kitchenham et al. [1] first suggested that software engineering should consider adopting evidence-based practice as pioneered in the field of medicine. The ideas were further considered in two subsequent papers, one discussing the implications for practitioners, the other discussing how evidence-based software engineering could be taught [2], [3].

Kitchenham et al. [1] defined the goal of evidence-based software engineering (EBSE) as:

To provide the means by which current best evidence from research can be integrated with practical experience and human values in the decision making process regarding the development and maintenance of software.

By analogy with medicine, Dybå et al. [2] identified evidence-based software engineering as a five step process:

1. Converting a relevant problem or information need into an answerable question.

2. Searching the literature for the best available evidence to answer the question.

3. Critically appraising the evidence for its validity, impact, and applicability.

4. Integrating the appraised evidence with practical experience and the values and circumstances of the customer to make decisions about practice.

5. Evaluating software development performance and seeking ways to improve it.

Although these papers were sympathetic to the idea of EBSE they pointed out some potential problems, for example:

- Human-centred software engineering experiments are skill-based. They cannot have the same objectivity as medical trials which can utilize double-blind protocols (i.e. experiments where neither the experimenter nor the subject knows which treatment the subject has received) to avoid subject or experimenter bias.

- Most software experiments are laboratory experiments which are difficult to generalize, whereas medical
controlled trials are field experiments where real patients receive real (or sometimes placebo) treatments.

- It is not clear who the "practitioner" is in the context of EBSE. Unlike medical practitioners, software engineers are not usually responsible for selecting and using appropriate methods, this is usually the task of senior managers. In our opinion, we should also regard authors of text books and developers of standards as "practitioners" in the sense that they are individuals that may want to make use of EBSE results.

- Facilities for searching digital libraries are not as good as those available to medics.

Also, EBSE relies on the existence of good quality empirical studies and the ability to aggregate the results of those studies. With respect to empirical studies in general, for over 30 years, Basili’s pioneering work on empirical software engineering has lead to an increasing acceptance of the need for empirical studies (e.g. ([4], [5], [6]).

Furthermore, since the mid 90’s there have been many initiatives aimed at improving the quality of empirical software engineering research. In 1995, the Empirical Software Engineering Journal was established. Several books on empirical software engineering were published ([7], [8], [9]). Kitchenham et al. published a paper specifying guidelines for empirical studies in software engineering [10] and Jedlitschka et al. presented a set of guidelines for reporting experiments [11]. The International Symposium on Empirical Software Engineering (ISESE) was established in 2003 and merged with the Metrics Symposium in 2008 to form the Empirical Software Engineering and Measurement Conference. Since 2005, researchers at the Simula Research Laboratory have published a series of systematic literature reviews investigating the nature and quality of experiments in software engineering ([12], [13], [14], [15]). This suggests that one basic requirement for EBSE is achieved to some extent.

With respect to the ability to aggregate good quality research results, evidence-based practice usually adopts the systematic literature review methodology. This methodology was first developed in the field of medicine (see for example [16]), but has also been adopted by social scientists ([17], [18]). In 2004, Kitchenham summarized current medical guidelines and attempted to adapt them for software engineering research [19]. The software engineering guidelines were updated in 2007 [20].

We are currently undertaking a research project "Evidence-based Practice Informing Computing" (EPIC) aimed at assessing the use of evidence-based practice and systematic literature reviews in software engineering <http://www.ebse.org.uk>. The results of the EPIC studies are addressing some of the issues raised by the guidelines for example:

- What problems do novice researchers face when performing systematic reviews [21]?
- How can we determine the quality of software engineering empirical studies that include technology-centric as well as human-centric empirical studies ([22], [23])?
- Are complex search strings as suggested by medical standards more efficient than simple search strings?
- Are broad searches rather than targeted searches necessary in software engineering [24]?

This paper aims to introduce the concepts of Evidence-based Software Engineering and in particular the systematic review methodology. Section 2 describes systematic literature reviews. Section 3 presents a summary of the distribution of systematic reviews across the SWEBOK categories [25] and presents our vision for the future of software engineering.

### 2 Systematic Reviews - Background

We start by explaining the basic methodology and rationale for systematic reviews. Then we discuss the different types of literature review that can broadly be called "systematic".

#### 2.1 Basic Methodology

Systematic reviews (sometimes called systematic literature reviews in software engineering to ensure they are not confused with inspection research) are a form of secondary study that assess the impact of individual empirical studies (referred to as primary studies). A systematic review involves several discrete activities. Existing guidelines for systematic reviews have slightly different suggestions about the number and order of activities. However, medical guidelines and sociological text books (e.g. [16], [17], [18]) are broadly in agreement about the three major stages in the process: Planning the Review, Conducting the Review, Reporting the Review.

The stages associated with planning the review are:

- Identification of the need for a review.
- Commissioning a review.
- Specifying the research question(s).
- Developing a review protocol.
- Evaluating the review protocol.

The stages associated with conducting the review are:

- Identification of research.
- Selection of primary studies.
- Study quality assessment.
- Data extraction and monitoring.
- Data synthesis.

The stages associated with reporting the review are:

- Specifying dissemination mechanisms.
- Formatting the main report.
- Evaluating the report.

We consider all the above stages to be mandatory except:

- Commissioning a review which depends on whether or not the systematic review is being done on a commercial basis.

Evaluating the review protocol and Evaluating the report which are optional and depend on the quality assurance procedures decided by the systematic review team (and any other stakeholders).

The stages listed above may appear to be sequential,
but it is important to recognize that many of the stages involve iteration. In particular, many activities are trialled during the protocol development stage, and refined when the review proper takes place.

2.2 Goals and Rationale

Systematic reviews aim to search for, and aggregate, all relevant information on a specific research topic, where in the context of evidence-based software engineering, "information" implies empirical evidence. (However, there is no reason not to use a systematic approach if you need to aggregate other forms of research.)

The use of the term "all relevant" qualifying "information" implies that all primary studies addressing the research topics should be included in the review. However, many researchers prefer to restrict the aggregation process to "best quality" research. For example, in the context of software engineering research, researchers may identify many "lessons learnt" papers that were conducted without using any well-defined research methodology. Such studies may be excluded from any aggregation of primary study results due to poor quality (see [26] for a discussion of how to improve experience reports).

The critical issue for systematic reviews compared with conventional reviews is that a systematic review uses a defined methodology that aims to ensure that the review is both fair and seen to be fair. In particular, systematic reviews aim to be:

- **Open** i.e. all the review procedures are reported initially in a study plan (referred to as a protocol) and in the final report of the study (where distortions from the protocol need to be reported).
- **Unbiased** i.e. (as far as possible) all relevant primary studies are included and the results of the included primary studies are fairly aggregated.
- **Repeatable** i.e. (as far as possible) the review could be replicated by other researchers. Note, however, when searching digital libraries, even if electronic search strings are specified, search results may not be completely repeatable.

The goals support one another. The requirement for an open methodology supports the goal for unbiased search and selection of primary studies and helps to support the requirements for repeatability. The detailed procedures used in a systematic literature review also support these goals. For example, the use of a study protocol not only improves study conduct but also reduces the opportunity for researcher bias, since stating in advance what sources will be searched and how avoids relying on the personal knowledge of individual researchers. Furthermore, a protocol helps to ensure that the study is open and repeatable.

2.3 Systematic vs. Non-systematic Reviews

Kitchenham and Charters [20] and Brereton et al. [27] both make it clear that the rigour of the systematic reviews process means that it uses more time and effort than typical "expert opinion" based reviews where the included studies and the aggregation process depends on the expertise of the authors. However, results in other disciplines confirm that reviews may miss relevant papers. For example, Oakley and colleagues undertook two investigations of reviews, one related to health promotion for the elderly [28], the other related to anti-smoking education among young people ([29] reported in [17]). In the first case there were six reviews of differing rigour identifying a total of 137 studies, but only 33 studies were common to at least two reviews. In the second case there were two reviews identifying 27 studies of which only three studies were common to both reviews.

Missing relevant studies can lead to drawing incorrect conclusions even when undertaken by experts. For example, after an informal review, the Nobel Laureate Linus Pauling was convinced that mega-doses of vitamin C would protect against the common cold. However a systematic review contradicted this conclusion and identified that Pauling had not cited 5 of the 15 methodologically sound studies [30]. In addition, authors can be biased in their selection of papers to cite. Shaddish surveyed authors of over 280 articles in psychological journals and found studies were often cited because they supported the author’s argument, not because they were good quality studies [31].

In the field of software engineering, there are examples of systematic reviews that have overturned "common knowledge" or suggested we are not as well-informed as we think about our methods and tools:

- Jørgensen [32] investigated the results of studies that compared algorithmic cost estimation models with expert opinion estimates. Although cost estimation research for the past thirty years has been premised upon the inferiority of expert opinion estimates, he found no compelling evidence that algorithmic models are more accurate than expert opinion estimates. In fact, one third of the studies said algorithmic models were best, one third said expert opinion-based estimates are best and one third found no difference.
- For many years the original Standish Chaos report was taken as confirming the parlous state of software engineering with the majority of projects failing and significantly overrunning. As background to a study of Norwegian projects, Moløkken-Ostvold et al. [33] looked at all empirical studies that reported failure rates and over-runs. They found that the Standish report was completely out of line with other contemporary studies, which suggested average overruns in the region of 30%. This figure has not changed much in the last 20 years. The methodological problems with the original Standish report (including explicitly soliciting reports of failing projects) were the subject of another study [34].
- Dybå and Dingsøyr recently reviewed holistic empirical studies of agile methods i.e. the results of applying an integrated set of agile methods [35]. In contrast, Hannay et al. have reviewed empirical studies of pair-programming [36]. Both studies suggest we know a lot less about agile techniques than we might think. Dybå and Dingsøyr highlight the general lack of trustworthy empirical studies, particularly related to management methods – they only found...
one study investigating SCRUM. Hannay et al. found limited evidence that agile methods exhibit greater quality and are faster than conventional methods but are less productive. However, all of these effects were small and detailed meta-analysis casts some doubts on the reliability of the effects.

Our own study of the TAM (Technology Acceptance Model) draws upon 73 primary studies [37]. This has demonstrated the lack of objective measures for technology use in many studies showing that studies of the TAM often measure perceived use rather than actual use and also that some of the key components of the TAM are not good predictors of actual use.

A number of mapping studies, a form of secondary study that seeks to scope our empirical knowledge about a topic (see Section 2.4), have been undertaken as part of our own studies. While in other disciplines these form a useful start point for a fuller SLR, when performed for software engineering they have largely revealed the major gaps in our empirical knowledge. Examples have included UML [38], object-oriented design [39] and design patterns [40].

### 2.4 Types of Systematic Review

A standard systematic review is driven by a very specific research question that can be answered by empirical research, for example "Are algorithmic cost models more accurate than expert judgement-based estimates?" This research question drives the identification of appropriate primary studies, informs the data extraction process applied to each included primary study, and determines the aggregation of the extracted data. This type of secondary study contributes most to evidence-based software engineering. If the research question is of interest to industry, it provides an unbiased summary of current empirical knowledge that can support the identification of good practice guidelines.

There are however two other types of rigorous secondary study found in software engineering journals that use a similar methodology to a standard systematic review but have slightly different goals:

- **Research method studies**. This type of secondary study reviews a set of primary studies with the aim of assessing the methods by which a discipline undertakes its research, see for example [41]. Such a study uses a systematic process for selecting primary studies (in this case a random sample from all papers published in a specific set of publications in a specified time period). Thus, repeatability rests on a statistical argument. Each primary study is classified and the results are aggregated by enumerating the percentages of primary studies in various categories. Although the basic secondary study process is consistent with a systematic review, such studies do not aim to address specific empirical research questions. They are not usually of direct interest to researchers concerned with software engineering topics nor to industry practitioners.

- **Mapping studies**. This type of secondary study reviews a specific software engineering topic and classifies the primary research papers in that specific domain. The research questions for such a study concern which sub-topics have been addressed, what empirical methods have been used, and what sub-topics have sufficient empirical studies for a more detailed systematic review. Mapping studies are of great potential importance to software engineering researchers. For example, two particularly influential mapping studies looked at cost estimation studies [42] and software experiments [12]. These studies catalogued the primary studies and their results were used by later more detailed systematic reviews ([13], [14], [15], [43]). The subsequent studies were easier to do because the original studies had already identified the relevant literature.

Although at the extreme, mapping studies and systemic reviews have rather different goals, there is often an overlap. Some systematic reviews include a classification system to organize relevant literature followed by a more detailed description of the research within each category. The important difference is that a conventional systematic review makes an attempt to aggregate the primary studies in terms of the research outcomes and investigates whether those research outcomes are consistent or contradictory. In contrast, a mapping study usually aims only to classify the relevant literature.

Finally, a meta-analysis is also a type of systematic review. Basically, a meta-analysis is an add-on to a conventional systematic review. It relies on systematic review procedures to search and select the relevant primary studies and define the data extraction process, but uses statistical methods to aggregate primary study outcomes rather than simple tabulation or narrative descriptions (see for example [36]).

A systematic review assessing empirical evidence related to a specific software engineering issue can save time and effort for researchers and practitioners. If it identifies areas where no further research is necessary, there is the opportunity to give clear-cut advice to practitioners and to provide a reliable empirical basis for our international standards and text books. If it identifies an import research question where empirical research is sparse, it provides a clear justification for more detailed primary research.

A systematic mapping study is usually of most relevance to researchers. A problem with some mapping studies is that they only present summary results without the full list of categorized primary studies (e.g. [38], [39], [40]). However, a mapping study which includes all the references provides an excellent starting point for other researchers.

Finally we must emphasize that systematic reviews are not meant to be solely of interest to researchers. Results of good quality systematic reviews provide an unbiased assessment of the empirical evidence supporting proposed software engineering methods. Such studies should provide a welcome dose of realism compared with non-peer reviewed "white papers" prepared by vendors and consultants with vested interests in selling products and services. Industry is often critical of academic research, but a significant benefit of high quality academic research is its impartiality. Systematic reviews provide a framework for providing impartial summaries of empirical results which are
the basis for evidence-based practice (see for example [44], [45]). Furthermore, researchers have a responsibility to make their results accessible to industry. They should ensure that the results of systematic reviews are explained (whenever possible) in terms of their implications for practice.

However, evidence-based practice also depends on practitioners appreciating the need for high quality evidence and understanding the role researchers have in the development (via primary studies) and aggregation (via secondary studies) of empirical evidence. This would be helped if more of our text books and international standards were based on evidence rather than simply "expert" opinion.

### 3 Systematic Reviews of Software Engineering

Within software engineering, the SWEBOK [25] has become established as an authoritative reference, drawing widely upon expert opinion and experience, and providing an overview of the topics and practices that currently form the knowledge-base for software engineering research and practice. In this section we use the structure of the SWEBOK to summarize existing software engineering systematic reviews.

We recently published the results of a mapping study aimed at identifying software engineering systematic reviews [46]. The goal of the study was to identify how many systematic reviews had been published, what research topics were being addressed, and the limitations of current systematic reviews. For this study we used a manual search of a targeted set of 13 conferences and journals published during the period January 1 2004 to 30 June 2007. The sources were selected because they were known to include either empirical studies or literature surveys, and had been used as sources for other systematic mapping studies (e.g. [12], [41]).

We found 20 studies (two of which were found by consulting known researchers or their web sites). We have recently updated this study using a broader automated search strategy (searching 6 electronic sources) from January 2004 to July 2008 [24]. We found an extra 12 systematic reviews in the period January 2004-June 2007, and 19 systematic reviews in the period July 2007-June 2008, plus 1 extra review published in July 2008. In addition, the Information and Software Technology journal has just published an online Virtual Special Issue of systematic reviews (linked by a banner to the IST home page, <http://www.elsevier.com/wps/find/journaldescription.cws_home/525444/description>). This special issue includes eight recent systematic reviews of which only one was included in our study (because it was published prior to June 2008). Thus, in total we have found 59 systematic reviews of which 34 addressed software engineering questions rather than mapping general research trends or software engineering topics. Table 1 shows the current state of evidence-based knowledge when mapped on to the structure of the SWEBOK. The total number of relevant reviews totals 32 not 34 because one review [47] was an intentional replication of another (i.e. it addressed exactly the same research question as another) and another review addressed the motivation of individual software engineers [48] and so was outside the scope of the SWEBOK which includes software project management issues but not general management issues.

Because Table 1 is coarse-grained at the level of SWEBOK chapter headings, it can be slightly misleading. In particular, although the largest number of secondary studies is reported under Software Engineering Management, 8 of these are essentially under just one of its sub-headings (Effort, schedule and cost estimation). More important though, as indicated above, the results from a number of these studies contradict established practices and opinions. Under Software Engineering Process, three papers relate to Software Lifecycle Processes and three relate to Process Assessment Models. Furthermore, our results confirm that even at the level of subsections, the SWEBOK is itself very coarse grained in places, so there can be many different systematic reviews that apply to a single subsection.

The number of systematic reviews is encouraging. However, overall the coverage of the SWEBOK is limited, although this reflects a lack of basic empirical studies as well as a lack of systematic reviews.

There is a very real need to provide a better basis for making IT/Computing related decisions than simply using expert judgement, however well this is codified ([49], [50]). Questions about the effectiveness of large-scale government IT procurement indicate that there is also a need to provide a firmer basis for such projects wherever possible.

We believe that the use of the evidence-based paradigm in software engineering provides a means to address these needs. Furthermore, a goal of an empirical software engineering BOK would provide an ideal framework for researchers and practitioners. It would provide a basic research agenda for researchers undertaking both primary and secondary studies, as well as a structure for making results available to practitioners (engineers and managers), standards and text book authors, and students.

### Acknowledgements

This study was funded by the UK Engineering and Physical Sciences Research Council project EPIC/E046983/1.

### References


4. V.R. Basili, M.V. Zelkowitz. The software engineering laboratory: Objectives, Proceedings of the fifteenth...
### Table 1: Distribution of Existing Software Engineering Systematic Reviews.

<table>
<thead>
<tr>
<th>SWEBOK Chapters</th>
<th>Title</th>
<th>Number of Sections</th>
<th>Number of Sections with SLR coverage</th>
<th>Total number of SLRs</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Software Requirements</td>
<td>28</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>Software Design</td>
<td>30</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>Software Construction</td>
<td>14</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>Software testing</td>
<td>68</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>Software Maintenance</td>
<td>28</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>Software Configuration Management</td>
<td>28</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>Software Engineering Management</td>
<td>24</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>Software Engineering process</td>
<td>20</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>10</td>
<td>Software Engineering Tools and Methods</td>
<td>13</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>Software Quality</td>
<td>24</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>277</td>
<td>19</td>
<td>32</td>
</tr>
</tbody>
</table>


Software Project Success: Moving Beyond Failure

Darren Dalcher

Success and failure in software projects appear to be difficult to define. While there is a consensus around the prevalence of project failure, new projects seem destined to repeat past mistakes. This paper tries to advance the discussion by offering a new perspective for reasoning about the meaning of success and the different types of failures. In order to court project success, practitioners need to rise beyond a fixation with internal parameters of efficiency to recognize the role of quality in bringing about the effectiveness required to secure project success. The paper begins by discussing project failure surveys and the impact of project constraints before offering a richer model that identifies the crucial role of quality in securing future success. The paper concludes by introducing a series of mini-case studies that help in making sense of success and failure and in particular highlight the interplay between the four levels of success.

Keywords: Effectiveness, Efficiency, IT Project Failure, Levels of Project Success, Project Failure, Project Management, Project Outcome, Project Quality, Project Success, Quality, Software Project Management, Software Projects, The Triple Constraint.

1 Starting with Project Failure

Popular computing literature is awash with stories of IS development failures and their adverse impact on individuals, organizations, and societal infrastructure. Indeed, contemporary software development practice is regularly characterized by runaway projects, late delivery, exceeded budgets, reduced functionality, and questionable quality that often results in cancellations, reduced scope, and significant re-work cycles [1].

The net result is an accumulation of waste, typically measured in financial terms. For example, in 1995, failed US projects cost $81 billion, with an additional $59 billion overspent, totalling $140 billion [2]. Jones contended that the average US cancelled project was a year late, having consumed 200% of its expected budget at the point of cancellation [3]. In 1996, failed projects alone totalled an estimated $100 billion [4]. In 1998, 28% of projects failed, at a cost of $75 billion, while in 2000, 65,000 US projects were reported to be failing [2].

IT failure is not a new phenomenon. Thirty years ago a GAO report in the US [5] showed that there were serious problems associated with the development of software. Less than 2% of the total value of contracts could be used efficiently as delivered and a further 3% could only be used after changes. The rest of the projects had the software delivered but never successfully used; the software paid for but not delivered; or the software used but extensively re-worked or later abandoned. Moreover, the first edition of the best-selling book on software engineering, The Mythical Man-Month: Essays on Software Engineering, tells the story, from the perspective of the project manager trying to stabilize the project, of a huge IBM software project with major cost and schedule delays which teetered on the brink of disaster for a number of years [6]. Indeed, the OS360 project came close to bankrupting IBM.

Failures tell a potentially grim tale. In 1995, 31.1% of US software projects were cancelled, while 52.7% were...
completed late, over budget (cost 189% of their original budget), and lacked essential functionality. Only 16.2% of projects were completed on time and within budget; only 9% in larger companies, where completed projects had an average desired functionality of 42% [2]. The 1996 cancellation figure rose to 40% [2] before improving to around 15% in 2002 (see Figure 1). However, the most recent figures reveal that the current failure rate is 24% [7][8].

While the scientific approach used by the Standish Group has been challenged over the methodology adopted and its rigour, the figures provide a useful baseline related to project failures. Other studies appear to confirm the high failure rates. For example, in 2004 a PriceWaterhouseCoopers study surveyed 10,640 projects and revealed that only 2.5% of companies achieve budget, scope, and schedule targets in all their projects. More recently, McManus and Wood-Harper discovered that only one in eight IT projects can be considered to be truly successful [9].

2 Beyond Simple Success Measures

The relationship between success and failure is not clear. Some view the relationship as a binary function so that a project is either successful or not. Research by McManus and Wood-Harper describes failure as “those projects that do not meet the original time, cost and requirements criteria”. The Standish Group makes a further distinction between failed projects and challenged projects. Failed projects are cancelled before completion, never implemented, or scrapped following installation. Challenged projects are completed and operational projects which are over-budget, late, and with fewer features and functions than initially specified. Successful projects however are completed on time, on budget, with all specified features. Figure 1 shows the relationship between successful, challenged and failed projects. Observing the Standish figures over the past 15 years there would appear to be a rough rule of thumb suggesting a split of 25% of projects being successful, 50% being challenged, and 25% failing.

The Oxford Dictionary defines success as a favourable outcome; doing what was desired or attempted; the accomplishment of an aim or purpose; or the attainment of wealth or fame or position. Failure is broadly defined as lack of success supporting the idea of a binary relationship. In an attempt to make further sense of the relative positions of success and failure software surveys have clearly found it useful to introduce the idea of partial failure (or challenged projects) as an intermediate position between success and failure, potentially indicating dissatisfaction with a two state explanation. Indeed many project outcomes do not fall directly into either category.

The above studies define success as meeting all the criteria associated with the budget, schedule and functionality, or scope; with failure viewed as a failure to meet all of the same criteria. This implies that if a project is finished on time, within budget and to quality (or covering the anticipated scope) it can be viewed as successful. Conversely, failing to meet any of the criteria will deem it a failure. The view is predicated on the traditional measures applied in project management and generally known as the triple constraint, the golden triangle, or the iron triangle.

Traditional theory holds that optimizing the three criteria will result in ideal performance on the project. Typical projects require a balancing act between the triple constraints of budget, schedule, and scope (see Figure 2). Trade-offs and adjustments are therefore made by restricting, adding to, or adjusting the cost, time and scope associated with

![Standish figures 1994-2008](image_url)
a project. Indeed the traditional triangle in project management is said to be concerned with finding a balance between cost, time and scope. For example, the more that is requested in terms of scope (or arguably even the performance or the quality), the more it is likely to cost and the longer the expected duration. If the client needs to have a certain performance delivered very rapidly, this will increase the cost due to the need to work faster and have more people involved in the development. The more features expected from a system, the higher the cost and the longer the expected duration. Conversely, if the costs need to be kept to a minimum, one may need to consider the essential performance, or the overall scope, and compromise there [10].

Many managers quickly discover that the triangle is not flexible. The three factors are closely entwined and project managers are expected to balance the **what** (scope) with the **when** (schedule) and the **how much** (budget). In practice, performance and scope are often determined prior to the project. Moreover, project managers often inherit the overall budget from the contracting activities that may even have imposed a fixed-price contract structure. A fixed overall budget may also exclude typical remedies like the hiring of specialists and the addition of human resources. The only remaining scope for leverage is in the schedule. However, this may also be imposed on a project through a fixed date for delivery with little regard to the complexity of the intended system or the risks it embodies. Once both budget and schedule are fixed, there appears to be little scope for compromises and tradeoffs.

The three factors thus play a key part in determining the degree to which a project is challenged (or even deemed a failure), yet they may be uncontrollable by the project manager. Indeed, Capers Jones observed that the most common constraints encountered are: fixed delivery dates; fixed-price contracts; staffing or team size limitations; and performance or throughput constraints [11] i.e. fixed time, price, staffing level, performance and scope. Many managers are thus looking to control other factors that may alter the outcome of the project, especially as the constraints often occur in concert. Measuring success on the basis of pre-established parameters that cannot be adjusted is therefore of limited value.

Before addressing additional factors in the next section it is also useful to point out that the artefacts of projects interact with organizations, customers, stakeholders, and other systems. Their impacts, regardless of whether or not they are delivered on time, can be crucial and perhaps even fatal in financial or real terms. Dalcher reports on the impact of an ambulance despatch system that was delivered to the users (at the third attempt), yet failed in action subsequently, potentially leading to loss of life [12].

Another illustration is a UK disaster which followed an earlier, yet unrelated, failure. The delay in introducing the Nirs2 system into the Inland Revenue beginning in 1995 meant that additional backlogs were building up. The backlogs caused the Inland Revenue to stop sending reminders to up to a third of the UK working force warning them that they needed to top up their national insurance contributions. As a result around 10 million people face a state pension shortfall. The impacted party includes the lowest paid workers in the UK. While the backlog resulted from a delayed system that itself cost tax payers millions of pounds, the additional loss will be borne by individuals and only count as a hidden backlog indirectly stemming from another failure. The true cost to individuals is likely to be £15 billion and the hardship that ensues as a result [13].
The success, and failure, of IT projects therefore cannot be delimited by a simplified set of factors and constraints associated with the delivery effort.

3 Making Sense of Project Success

Project success is a rather nebulous concept and the focus on the triple constraint can be too limiting. Indeed, Linbeg asserted that a whole new theory of project success is needed [14]. Pinto and Slevin noted that success combines issues related to the project itself with issues related to the client [15]. Moreover, software developers and systems analysts have recognized long ago that user involvement, satisfaction and buy-in are crucial to the success of IT projects. Prototyping and user-driven approaches were developed to maximize the potential for satisfaction for various stakeholders and thus increase the likelihood of user acceptance of the ultimate system.

Baccarini identifies the need to distinguish between project management success and the success of the product which entails dealing with the effects of the project’s final delivered product [16] thereby allaying the need to define a further dimension concerned with client expectations. Given that the product will be used by the client, there is a degree of similarity between the dichotomies put forward by Pinto and Slevin and by Baccarini. Cooke-Davies likewise makes a distinction between the focus on performance and the need to look at the success of a project [17].

Having multiple categories would suggest that it is possible to be successful in some areas and not successful in others. It thus makes it possible to understand mismatches between the different criteria and groups. Moreover, it implies that the traditional triple constraints of costs, time and performance only reveal part of the picture. In other words, it may be possible to maximize the traditional criteria and yet deliver a product that is not valued by the users. Likewise it is also possible to exceed the traditional criteria but deliver a product that is valued and adopted by the user community, despite exceeding the budget or the schedule.

4 Towards Multiple Levels of Success

Acknowledging the role of modern business in all projects makes a case for adopting a wider perspective of success that encompasses the various levels and ideas explored in this discussion. A tabular representation of four levels of success is offered in Table 2 [19].

Level 1 represents project management success and is thus concerned with internal efficiency and performance measurement and optimization at the project level through the tracking of the budget, schedule and scope, or functionality, parameters. Level 1 success is therefore to do with project delivery against the constraints or measures imposed on the project.

Level 2 is focused on the overall effectiveness of the project through the lens of what is actually being delivered. Success is measured through the quality and acceptability of the output that has been delivered. The benefits of the projects and the achievement of the objectives are thus assessed in terms of the satisfaction of the customer and the different stakeholder groups. Level 2 success reflects the acceptability of the resulting artefact and the benefit that it delivers, the degree to which it is used, the qual-

**Table 1: Crucial Issues in Failures.**

<table>
<thead>
<tr>
<th>Crucial Issues in Failures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relationship management</td>
</tr>
<tr>
<td>Trust</td>
</tr>
<tr>
<td>Communication</td>
</tr>
<tr>
<td>Management of expectations</td>
</tr>
<tr>
<td>Politics</td>
</tr>
<tr>
<td>Risk management</td>
</tr>
<tr>
<td>Escalation</td>
</tr>
<tr>
<td>Contract management</td>
</tr>
</tbody>
</table>

© CEPIS
ity built into it, the match with project objectives and requirements, the relationship with the various stakeholder groups, and the overall impact on the customer.

Level 3 is centred on business efficiency which is assessed through the creation and delivery of internal value. The outcome of the project contributes to business success through the satisfaction of business objectives that have been realised. Success equates to maximization of financial and business efficiency measures, such as sales, profits or ROI (Return On Investment), as well as delivered value measures.

Level 4 is forward looking and opportunistic and enhances the business horizon by projecting future gains and opening new avenues, capabilities, skills and markets. Strategic opportunities require a continuous and long-term approach that seeks to derive not just immediate benefit but also to maximize opportunities for cornering the market, creating killer applications, and building the potential for self-enhancing positive feedback loops to secure future growth. Level 4 success is achieved through the realization of new opportunities and harnessing of new potential. It may include new uses or ideas that were not originally considered as well as the development of new competence or capability.

5 Mapping Success

It is interesting to note the horizon of activity for each of the levels of success. Level 1, Project management success is concerned with the execution of the project itself based on the performance against internally established constraints. Success at this level is determined upon delivery of the project, or even through the incremental delivery of

<table>
<thead>
<tr>
<th>Focus vs. Output/Outcome</th>
<th>Output</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>External Focus</strong></td>
<td>Level 2: Project success</td>
<td>Level 4: Future potential success</td>
</tr>
<tr>
<td></td>
<td>Quality of deliverables and other outputs (stakeholder view)</td>
<td>Business potential and wider implications (to shareholders)</td>
</tr>
<tr>
<td><strong>Internal Focus</strong></td>
<td>Level 1: Project management success</td>
<td>Level 3: Business success</td>
</tr>
<tr>
<td></td>
<td>Internal measures and constraints</td>
<td>Internal business value realized following project investment</td>
</tr>
</tbody>
</table>

Table 3: Focus vs. Output/Outcome
partial targets. It is primarily concerned with the task of project management. This is what most failure surveys assess and therefore where most failures are observed. Level 2 success is more deeply entwined with the technical activities resulting in the product or deliverables; indeed this is where quality provides the key to the assessment of success. Both levels can be said to be output driven as they look at the complementary aspects of technical action and management within established and imposed constraints. Level 2 success can extend to cover the entire operational life of the project output. After all, delivering a bridge which stands for one year before collapsing is far from being a mark of either quality or success.

The higher levels of success demarcate the shift from efficiency and output, to outcome and value as more strategic considerations come into play. Indeed while Levels 1 and 2 emphasise the output of the project, Levels 3 and 4 deal with the outcomes. The main distinction is that outputs occur as a result of a process as they are specified deliverables (that are delivered within time, cost and quality). Outcomes are the effects of change, and how it can deliver value for the entire business often beyond the scope of the original project. This relationship is depicted in the Table 3.

The higher levels of success demarcate the shift from efficiency and output, to outcome and value as more strategic considerations come into play. Indeed while Levels 1 and 2 emphasise the output of the project, Levels 3 and 4 deal with the outcomes. The main distinction is that outputs occur as a result of a process as they are specified deliverables (that are delivered within time, cost and quality). Outcomes are the effects of change, and how it can deliver value for the entire business often beyond the scope of the original project. This relationship is depicted in the Table 3.

Encouraging long-term thinking is important from a strategic perspective. It also fits with the need to deal with extended life cycles and considers deployment, extended use and decommissioning of artefacts alongside benefiting from new opportunities and market possibilities. Moreover it also chimes with the idea of viewing software development as the development of a continuous service (implying long term relationships and strategic concerns) rather than the delivery of a single artefact. It is worth noting that, while Levels 1 and 2 are primarily concerned with the delivery of a single project, the remaining levels look beyond a single delivery view using a more strategic lens.

A further important distinction is the separation between efficiency and effectiveness. Project managers and software developers have shown a tendency to focus on efficiency and its implications, as is reflected by the obsession with failure studies. However quality solutions emerge from consideration of effectiveness. Efficiency is essentially viewed as productivity metric, as it is concerned with doing things right and by implication with following procedures and processes, adhering to constraints or achieving with minimum resources. Effectiveness on the other hand, deals with doing the right things and is therefore a quality metric; the ends to effectiveness’ means. The relationship is depicted in Table 4.

**6 Effectiveness: the Case for Quality**

Failure studies and surveys seem to focus on criteria concerned with the efficiency of projects, while ignoring the effectiveness aspects, and thus sidestepping the major issues associated with quality. Indeed, many papers on quality seem concerned with the track record of projects as a measure of quality. However this is not the case as the triple constraints measures the ability to predict deadlines when uncertainty is highest and stick to them. This is not a measure of quality and is therefore addressed as Level 1 success. To attain project success one needs to relate to the quality aspects and perspectives related to the effective-
ness of the project.

Contemporary understanding of quality implies moving away from the popular definition associated with high quality towards meeting the requirements that were specified for the project and that are essential for the users. It is now clear that quality is the main concern whenever we move beyond efficiency to consider effectiveness. The key to project success is being able to measure and indeed attain a sufficient quality level to ensure the project ultimately becomes a success. Project success requires consideration of the following [10]:

**Quality of design** introduces the concept of designers proactively deciding the level of quality that they consider is required. The level of quality therefore defines the characteristics specified by the designers, such as the type and grade of materials and their tolerances and the performance specifications.

**Quality of conformance** refers to the degree to which the design specifications are followed during manufacturing. Many other definitions of quality simply focus on conformance to requirements, which means that the project’s processes and products meet the specified requirements.

**Fitness for purpose** means that a product can be used for the purpose it was intended. This is typically seen as being more rigorous than “fitness for use”.

Quality and success are judged by different stakeholders in different ways, utilising different criteria, over different timescales [20][21][22]. Recently, there has been a tendency to let the customer define quality. The Kodak organization defines quality as “those products or services that are perceived to meet or exceed the needs and expectations of the customer at a cost that represents outstanding value” [23].

The interesting point to note with this definition is how the customer viewpoint impacts on a project: a project must take great care that it accurately defines customers’ needs and expectations, and the ultimate power of deciding on quality is given to the customers. So with this definition, conformance to requirements is not necessarily sufficient—the customer must be satisfied with the resulting product or service. Further, in order to maintain the satisfaction of customers and their loyalty and to ensure higher level success, products need to be revised and adjusted to reflect shifting needs and expectations (as well as market trends and the competition). So maintaining quality becomes a continuous process of product (and process) improvement [10].

Over the years, several different views of quality have been employed, allowing project managers to select from a contingency of approaches and perspectives including the following:

- **Quality as a product-based quantity**: this is the traditional view of quality. The assumption is that quality is related to the content of the product

- **Quality as a user-based view**: quality is based on the values of the users. Such a view will therefore encompass the user’s ideas through the notion of fitness for purpose and conformance to requirements. Initially this was viewed as a static value that had to be extracted prior to embarking on the process of development, but it is now understood to be an evolving set of values and preferences

**Quality as a specification**: this view is derived from the manufacturing industries. The assumption is that a clear (technical) specification of the product exists or can be obtained. Quality can therefore be determined as conformance to this formal specification

**Quality as a value-based approach**: the value-based perspective acts as a composite of the last two views. Quality is assumed to equate to what the user wants at an acceptable price and while conforming to an exact specification at an acceptable cost. Quality can thus be equated with value to the user (justified in terms of manufacturing costs)

**Quality as a transcendent property**: quality can be equated with some kind of innate excellence. The exact parameters cannot be defined precisely. It is also difficult to impose tests to ascertain achievement since quality is felt rather than measured

**Quality as a continuous property**: the modern approach views quality as the evolving satisfaction level of the users. The view is that change will force adjustments and that all aspects of the system, including quality, must be considered to be dynamic.

In general there appears to be an increasing emphasis on customer- and stakeholder- involvement which hopefully leads to project success. Modern project quality management emphasises the usefulness and acceptability of a product to its users. The satisfaction levels that the project solutions offer to clients and users are used as the measures for project success (Level 2 success). A further illustration of quality in the context of success is offered through the short case descriptions that follow.

### 7 Illustrative Examples

To highlight the distinctive features of the levels and the differences between them, it might be instructive to focus on thumbnail sketch examples from a range of sectors.

**Story 1**: The operation was successful but the patient died. Level 1 success—Level 2 failure

This paper described a number of failure surveys that focus on project management failure (i.e. the inability to meet time, cost and scope criteria). Project management success is no guarantee of project success as many targets are assigned arbitrarily at an early phase. For example, the third attempt to deliver a working ambulance dispatch system for London was delivered by the agreed deadline (Level 1 success), but stopped working a few days later, resulting in potential loss of life [12]. In IT projects this is typical of a project delivered on time and within budget covering the agreed scope which is ultimately never used by the users.

**Story 2**: The Millennium Dome

The project to deliver a dome-shaped building to house a yearlong exhibition to celebrate the millennium had to be available in time for the new Millennium, the building itself and the infrastructure enabling Londoners to experience the exhibition were just about finished on time, but following an unexpected injection of additional funds. On
opening night many of the exhibits were not functioning and dignitaries were left to queue outside for hours. Predicted visitor numbers exceeded one million per month but in practice about a third of the expected visitors turned up. The exhibition had to be kept open for a further year (in clear violation of the stated intention) to try and recoup some of the costs, while the entry fee was halved to attract visitors. Following the end of the exhibition the site was mothballed at a cost of £190,000 a year adding to the accumulated losses. However, once the sale was finally concluded, the renamed O2 centre became the biggest and most successful sports and entertainment arena in Europe. Level 4 success through innovative use of the structure thus managed to make up for the earlier short term disappointments (albeit in the hands of a new owner).

Story 3: The Sydney Opera House

An even more heralded failure which clearly failed in terms of Project Management, Project and Business. The Sydney Opera House came in at 14 times over budget, a clear project management failure. The building was unsuitable for its original purpose as the acoustics made it impossible to have concerts inside the building. However the building has become an icon and is considered to be an architectural marvel. It is attracting tourists from all over the world and generating revenue, not least for the entire city. The revenue is not generated under the original intention but the new potential has been utilized to the full. Interestingly, the building was not fit for purpose (and hence was not of acceptable quality), yet it managed to generate a new purpose – for which it was fit enough.

Story 4: Project Orion

A massive effort to develop Kodak’s new Advantix photographic system was considered a big success on completion. The product was selected by Business Week as one of the best new products of 1996 (suggesting Project success). It also won the Project Management Institute International PMI Project of the Year award, making it a Project management success. The only problem was that Kodak failed to anticipate the accelerating switch to digital photography which made the product redundant. A successful output that won multiple awards was thus destined to become a failure as an outcome. In terms of quality the resulting product was an award winner but at the wrong end of the utilization and relevance curve, just as the technology slid out of fashion.

These brief vignettes highlight the complexity of success and the interplay between the different levels involved. Success is never simple, and the mini cases help shed further light on the rich, interconnected and intricate (and sometimes temporary) nature of success.

8 Conclusion

Project failures have highlighted the need to improve IT software project practice. Many of the studies and surveys focus on project management success (or failure) which can be described as a sub-set of internal efficiency measures and imposed constraints ignoring the impact on the project and the business. In order to improve project performance we need to look beyond such measures and focus on project success, an area concerned with the effectiveness and quality of the project output. Success is a complex and multi-layered concept which needs to be understood at different levels and in different timeframes. Indeed, the impact of success often extends beyond a single project. This paper offers a wider perspective which takes in a range of project success levels, thus enabling practitioners to move beyond the simplistic measures that continue to be offered. The success view determines actions and colours new developments. Increased attention to enterprise objectives and quality, rather than simply endeavouring to optimize correctness according to imposed constraints, can open a new dialogue about the needs of a profession seeking to fundamentally and essentially improve its track record and enable development practices to rise beyond the continuous obsession with failure. In order to overcome failure we must learn to appreciate success, and grow enough to look beyond the simplest manifestations of an imperfect practice.

References

Experiences and Advances in Software Quality


Software Measurement for Better Project and Process Quality

Christof Ebert

Software increasingly governs our world and our society. Since software is so ubiquitous and embedded in nearly everything we do, we need to stay in control. We have to make sure that systems and their software run as we intend - or better. Software measurement is the discipline that ensures that we stay in control. Software measurement applies to products (e.g., performance engineering), processes (e.g., productivity improvement), projects (e.g., estimation) and people (e.g., engineering skills). This article will introduce to software measurement in the context of achieving better quality for projects and processes.

Keywords: Empirical Software Engineering, Process Improvement, Project Management, Quantitative Management, Software Measurement.

1 Introduction

Human performance improvement is essentially unlimited. We are driven to try to go beyond what is thought to be our limits. The fastest time for the mile was 4 minutes 30 seconds in 1865, 4 minutes in 1954, and is around 3 minutes today (depending on when you read this introduction). Perhaps we can expect a three-minute mile during this century? Can you imagine how little runners would have improved if there were no stopwatch or measured track?

Unmeasured and unchallenged performance does not improve! Moreover, it will not improve if not fostered by best practices in the discipline. Software development is a human activity and is prone to continuous performance improvements. Software measurement is the approach to control and manage the software process and to track and improve its performance. You can only control what you observe and measure. The act of setting objectives and monitoring them is the way to guarantee receiving the expected results.

Software measurement is however not sufficiently used to keep projects on track and to improve organizations’ performance. Whether it is software engineering for a new embedded automation product, the development of a software application or the introduction of a managed IT-service, demand outstrips the resource capacity of an organization. As a result we see two things: First the acceptance of impossible constraints in time, scope and/or cost. And second, as a direct consequence of the first, we see an increase in rework and chaos, as well as budget overruns, delays and canceled projects. We will show here how software measurements contribute to project success. For more details we refer to the book "Software Measurement" [1].

2 Foundations of Software Measurement

The way software measurement is used in a software company determines how much business value that organization actually realizes. You can hardly imagine any software company operating at its maximum performance without knowing where it is and where it should go. It would be like a fast car in fog that needs to slow down below its capability due to not knowing where it is. Software measurements are used in the following ways:

1. Understand and communicate: Measurements help us understand more about software work products or underlying processes. It also allows us to evaluate a specific situation or (statistical) characteristics of software artifacts in order to make operational decisions leading to special experiences (e.g., project management, rules of thumb, assessments and analysis of situations).

2. Specify and achieve objectives. Measurements are key in identifying and specifying objectives. They ensure we stay on course and eventually reach those objectives. They are used to estimate, and/or forecast software characteristics in order to achieve and improve our understanding leading to improved general knowledge and guidelines (e.g., estimation formulas, development standards and general project profiles).

3. Identify and resolve problems. Measurements help us evaluate work products or processes against established standards and to define and measure specific attributes during the software life-cycle. The latter enable improvements in quality and/or performance (standardized size and complexity measurements). They help to identify, analyze and mitigate risks in order to optimize return on investment (ROI) in an organization’s business.

4. Decision-making and improvement. Measurements allow the monitoring, evaluation, analysis, and control of the business, project or, product or they also permit the con-
trol of process attributes for operational and strategic decisions leading to continuous improvement. This includes portfolio management, removal of the root causes of problems and process improvement.

Today, it is impossible to have a business process without measurements and periodic control. For some companies this is a legal obligation, such as provided by the Sarbanes-Oxley regulations. For all companies it is a simple question of audit and financial control. The measurement process is an inherent part of almost any business process (see Figure 1). Software engineering, as part of the product development business process, is therefore also in need of control and measurement. Software measurements include performance measurements, project control and process efficiency and effectiveness.

Measurements are management tools. Measurements must thus be governed by goals, such as reducing the number of project risks or improving test efficiency, as otherwise they simply generate data cemeteries. Figure 1 shows that independent of the process and its granularity, measurements are first established in terms of goals and measurement processes. They are then extracted in operational activities, where these measurements are evaluated for instance on whether the overarching goals will be reached, and definitive actions have to be executed to ensure that deviations from goals will be removed. This goal-oriented measurement process is called the E4–measurement process.

The E4–measurement process consists of four essential steps:

- **Establish** concrete objectives and the measurement and analysis scope and activities
- **Extract** measurements for the established objectives
- **Evaluate** this information in view of a specific background of actual status and goals
- **Execute** (carry out) a decision to reduce the differences between actual status and goals

The measurement process is governed by ISO 15939 (Software engineering – Software measurement process) [3]. By following this standard we can plan and implement a measurement process based upon best practices. ISO 15939 consists of two parts. First we need to establish the measurement program and prepare it within the project or organization. Then we execute measurements in each single project covering people and processes. We extract data, evaluate it and execute corrective actions – depending on the outcome of analyzing our measurements. These two parts, namely preparing and executing the measurement program, should be considered in all subsequent examples which we provide in this article. It is not enough to simply collect numbers and report them. There is a need for a full framework into which measurements can be embedded as a tool to support decision-making and to further drive the implementation of decisions.

To describe the use of measurement, we will introduce a simplified product life-cycle as it applies to software, systems and IT projects. The top portion of Figure 2 shows this simplified product life-cycle. It consists of archetypal phases as you find them in practically all product and service developments, like it software applications, internet software, middleware, enterprise and IT infrastructure, embedded systems, firmware or services. For each life-cycle phase the corresponding text box shows measurements which are relevant for making the project and product a success.

It is useless to extract information and then only record it for potential further use. If measurements are not used immediately, if they are not analyzed and evaluated, then the chances are high that the underlying data is invalid. If there is no pressure to have accurate measurements available, collection is done with little or no care. Where there is no direct use for the information, the information is worthless and the effort to collect it wasted.

### 3 Project Measurement

Do you have sufficient insight into your engineering and IT projects? If you are like the majority of those in IT and software companies, you only know the financial figures. Too many projects are often run in parallel, without concrete and quantitative objectives and without tracking where they are with respect to expectations. Project proposals are evaluated in isolation from ongoing activities. Projects are started or stopped based on local criteria, not by considering the global trade-offs across all projects and opportunities. Only one third of all software engineering companies systematically utilize techniques to measure and control their product releases and development projects [4].

Project management is the major application of software measurement. Software measurement is an absolutely necessary precondition for project success. The measurement goal is to master the project and to finish whilst meeting or exceeding commitments. We need a way to determine if a project is on track or not. There is a saying that "you cannot control what you cannot measure". Because there is little or no visibility of the status of and plan for projects, it is apparent that some common baseline measurements need to be implemented for all projects in an organization. Such core measurements provide visibility of how engineering projects status is performing against the plan, and allow the early detection of divergence and timely corrective action. Measurements reduce the likelihood and number of "surprises" by giving us insight into when a project is heading towards trouble, instead of discovering it when it is already there. Standardized measurements provide management with indicators to assist in controlling projects and the evaluation of performance in the bigger picture.

Project control answers a few simple questions derived from the following management activities:

- **Decision-making.** What should I do?
- **Focusing Attention.** What should I look at?
- **Performance evaluation.** How well am I doing?
- **Improvement tracking.** Am I doing better or worse than the last period?
- **Target setting.** What can we realistically achieve in a given period?
Experiences and Advances in Software Quality

Figure 1: The E4–Measurement Process with its Four Steps.

Figure 2: During each Phase of the Product Life Cycle, Dedicated Measurements are Extracted that Correspond to Established Objectives.
Planning. What is the best way to achieve our targets?

An initial set of internal project indicators for this goal can be derived from the Software Engineering Institute’s (SEI) core measurements [5] and measurement literature [1]. They simplify the selection by narrowing the focus onto project tracking and oversight from a contractor and program management perspective. Obviously, additional indicators must be agreed upon to evaluate external constraints and integrate with market data.

Here is our short-list of ”must have” project measurements:

1. Requirements status and volatility. Requirements status is a basic ingredient to tracking progress based on externally perceived value. Always remember that you are paid for implementing requirements, not for generating code.

2. Product size and complexity. Size can be measured as functional size in Function Points, or code size in lines of code or statements. Be prepared to distinguish according to your measurement goals with code size between what is new and what is reused or automatically generated code.

3. Effort. This is a basic monitoring parameter to assure you stay in budget. Effort is estimated upfront for the project and its activities. Afterwards these effort elements are tracked.

4. Schedule and time. The next basic monitoring measurement to ensure you can keep the scheduled delivery time. Similar to effort, time is broken down into increments or phases which are tracked based on what has been delivered so far. Note that milestone completion must be aligned with defined quality criteria to avoid poor quality/faults being detected too late.

5. Project progress. This is the key measurement during the entire project execution. Progress has many facets and should look to deliverables and how they contribute to achieving the project’s goals. Typically there are milestones for the big steps, and earned value and increments for day-to-day operational tracking. Earned value techniques look to the degree to which results such as implemented and tested requirements or closed work packages relate to effort spent and elapsed time. This then allows cost to complete and remaining time to complete the project to be estimated.

6. Quality. This is the most difficult measurement, as it is hardly possible to forecast accurately whether the product has already achieved the right quality level which is expected for operational usage. Quality measurements need to predict quality levels and track how many defects are found compared to estimated defects. Reviews, unit test and test progress and coverage are the key measurements to indicate quality. Reliability models are established to forecast how much defects need still to be found. Note that quality attributes not only measure functional quality but also relate to performance, security, safety and maintainability.

Projects typically aggregate information in similar manner to a dashboard. Such a project dashboard allows all relevant information related to project progress against commitments, including risks and other information summarized on one page. This is typically accessible online and is periodically updated. Examples of project dashboard information are actual milestones reached against the planned dates, or showing the earned value at a given moment.

Project dashboards provide information in a uniform way for all projects, and thus do not overload the user with different representations and semantics to be learnt and understood. They provide immediately available information for decision making. They help in examining those projects that underperform or that are exposed to increased risk. Project managers can look more closely and examine how they could solve such deviation in real time within the constraints of the project. All projects must share the same set of consistent measurements presented in a unique dashboard. Lots of time is actually wasted by reinventing spreadsheets and reporting formats, where instead the project team should focus on creating value.

A selection of the most relevant project tracking measurements is provided in Figure 3. There can be both direct measurements (e.g. cost) as well as indirect measurements and forecasts (e.g. cost to complete). Figure 3 shows a simplified dashboard used to track projects. It covers the major dimensions of milestone control, budget and cost control, quality level and earned value. These four dimensions give a rapid insight into progress in contrast to commitments and allowed resources. You realize in this dashboard the mixture of plan-driven tracking as we are used to from techniques such as PERT and the forecasting trends with defect predictions or earned value evolution.

These project control measurements are periodically updated and provide an easy overview of project status, even for very small projects. Based on this set of measurements, a small subset can be selected for weekly tracking of work products’ status and progress (e.g., increment availability, requirements progress, code delivery, defect detection), while others are reported periodically to build up a history database (e.g., size, effort...). Most of these measurements are actually byproducts from automatic collection tools related to planning and software configuration management (SCM) databases.

Figure 4 shows a customizable project dashboard that has all necessary information in one sheet, namely risks and open issues, budget and expense control, milestone control, earned value tracking, requirements and their respective implementation status, test planning and tracking and defects status. Built into the commercial eASEE PLM tools suite, it receives parts of its data from operational databases and others from the internal data backbone [6]. This ensures sufficient data quality to compare project status across all projects within a portfolio.

Make sure that numbers are consistent across these different hierarchies. Often aggregation hides insufficient data quality which is then only revealed when it is too late to improve those underlying processes. Figure 5 provides a practical example that we have established for a major Fortune 100 company. The corporate scorecard was the starting point and the necessary links to distributed operational tracking...
Experiences and Advances in Software Quality

Figure 3: Measurement Dashboard: Overview Measurements for Schedule, Cost, Quality and Earned Value.

data were established step by step with particular effort going into ensuring data quality by means of periodic reviews, governance and tool support. This small example also indicates that different processes such as corporate control, strategy management, portfolio management and project management are ultimately related. What goes wrong on one level must be visible on the next higher level – if it is beyond normal acceptable noise levels.

4 Process Measurement

Today, software is a major asset of many companies. Engineering investments are primarily spent on software development for the majority of applications and products. In our rapidly changing world, a company will only succeed if it continually challenges and optimizes its own performance. Engineering of technical products is currently undergoing a dramatic change. Ever more complex systems must be developed at decreasing cost and shortened time to market but keeping high quality standards. At the same time, competition is growing and the entry barriers to established markets are diminishing. The result is more competitors claiming that they can achieve better performance than established companies. An increasing number of companies are aware of these challenges and are pro-actively looking at ways to improve their development processes.

Development processes along the product life-cycle determine how things are done – end to end. They provide guidance to those who do and focus on what to do. Guidance means understanding and ensures repeatability. Focus means achieving targets both effectively and efficiently, without overheads, conflict or rework. Good processes are as lean and agile as possible, while still ensuring visibility, accountability and a commitment to results. Inefficient processes limit business opportunities and reduce performance because commitments can then not be kept and delivery is below expectations.

To remain competitive in terms of their software development, many companies are putting in place orchestrated improvement programs for their engineering processes.

Process improvement in today’s business is crucial to:
- Improve competitiveness (productivity).
- Expand markets.
- Extend market penetration.
- Better support customers.
- Address their real needs more effectively, to better manage the increasing variety of software assets.
- Be more profitable.

Software measurement is a necessary precondition of performance improvement. Process and product improvement must be combined with a strong focus on business objectives and measurements for following-up on change implementation. Otherwise, the risk is high that too much attention is focused on processes and not enough on what is essential for customers and shareholders. Improvements are directed by business objectives and need to be measured continuously in order to ensure focus, efficiency, and effectiveness. Emphasis is given to setting and dealing with quantitative objectives and thus making the progress of the im-
improvement initiative visible. To survive against today’s global competition with extremely low entry barriers in the software industry, it is important to continuously stretch targets and always keep pushing for improvements.

We have introduced the concept of objective-driven process improvement (ODPI) in order to focus processes on the objectives they must achieve. Processes are a means to an end and need to be lean, pragmatic, efficient and effective – or they will ultimately fail, despite all any amount of effort behind them.

Figure 6 shows this goal-driven relationship from business objectives to concrete annual performance objectives on an operational level to specific process performance measurements. Improvement goals cannot be reached if they
Experiences and Advances in Software Quality

Figure 5: Measurements Provide a Consistent View of All Engineering Projects, Product Lines and Business Units.

are not quantified and measured, or, as the saying goes, "managers without clear goals will not achieve their goals clearly".

Figure 6 shows concrete instances of objectives and measurements, such as improving schedule predictability or reducing cost. Naturally, they should be selected based on the market and business situation, the maturity and certainly the priorities within the projects.

The following example shows how objective-driven process improvement is translated into concrete actions:

**Step 1:** Identify the organization’s improvement needs from its business goals. These business goals provide the guidance for setting concrete engineering performance improvement objectives on a short-term basis.

Example: The business goal is to improve revenues and cash flows and to reduce the cost of non-performance due to schedule delays.

**Step 2:** Define and agree on the organization’s Key Performance Indicators (KPI’s). Such KPI’s are standardized across the organization and ensure visibility, accountability and comparability. Naturally KPI’s must relate to the business goals. Measurements should drive informed decision-making. They must be used for effectively communicating status and progress against the objectives set for the business, process or project. Measurements, both direct and indirect, should be periodically evaluated in conjunction with the driving objectives in order to identify problems or make decisions.

Example: The selected performance indicator is schedule predictability. It is measured as normalized delays compared to the originally agreed deadline. Schedule changes after project start are not considered in this measurement to avoid the situation where projects would argue that the delay is justified due to changing requirements. This definition is agreed with product managers and business owners to prevent them challenge engineering teams later.

**Step 3:** Identify the organization’s hot spots such as areas where they are feeling the most pain internally or from customers. Typical techniques include root cause analysis of defects and customer surveys.

Example: Average schedule overrun in the company is 45%. A root cause analysis of delays is performed. 40% of delays result from insufficient project management, 30% of delays come from changing requirements, 20% from supplier delays and 10% from other causes. Customer surveys underline that requirements are often included in the project which is perceived as offering higher flexibility but in fact experience shows that these are not critical requirements but rather nice-to-have features.

**Step 4:** Commit to concrete improvement objectives. These improvement objectives should directly address the above weaknesses and at the same time support the overarching business objective. Good objectives are commonly considered to be SMART:

- Specific (or precise).
- Measurable (or tangible).
- Accountable (or in line with individual responsibilities).
- Realistic (or achievable).
- Timely (or suitable for the current needs).
Experiences and Advances in Software Quality

These objectives are reviewed and approved by upper-level managers before proceeding further. This ensures that priorities are appropriate and that nothing relevant has been overlooked or misunderstood.

Example: Two improvement objectives are agreed, namely improving estimations and improving requirements development. The respective performance targets are agreed in a management seminar to achieve buy-in. Each project must have two estimates where the first is allowed to deviate by 20% and the second to deviate by 10%. Requirements change rate after project start has to be below 20% – except that customers pay a fixed fee calculated to be at least double the margins. Time-boxing and incremental development with prioritized requirements was then introduced to achieve those objectives. Requirements priorities were agreed across all impacted stakeholders from product management, engineering and marketing/sales before each new project start.

Step 5: Identify specific levers to start improvements and connect them to Return on Investment (ROI) planning. This is typically best done when using a process improvement framework such as CMMI [6]. This framework provides the necessary guidance on which best practices to apply and how processes relate to each other. Without the right levers, chances are high that objectives will not be reached.

Example: Focus will be on requirements development, requirements management, technical solutions, project planning and project monitoring and control. Project managers will be educated in project management techniques and negotiation skills.

Step 6: Perform a brief "gap analysis" of the selected process areas to identify strengths and weaknesses. A systematic look at weaknesses helps to focus limited engineering resources where it matters most.

Example: Requirements are collected rather than developed, requirements management satisfies basic needs for change management, engineering is too technology-driven and must be extended to capture business reasoning, project planning shows severe weaknesses in estimation and feasibility analysis, project monitoring and control has weaknesses in getting stakeholder agreement for changes.

Step 7: Develop a concrete action plan for the identified weaknesses. Avoid trying to change all of them at the same time. Use increments to subdivide bigger changes. Consider available resources and skills and get external support if you lack competencies, such as change management.

Example: The most urgent need is project planning. A dedicated one-month initiative is launched right away to install a suitable estimation method and train people on it. A tool for feasibility analysis is introduced in parallel because not much historic data was available. A history database is installed for a set of key project measurements. In a second phase, earned value analysis will be introduced. After three months requirements development is launched under the leadership of product management.

Step 8: Implement the changes to operational projects and measure progress against the improvement objectives that have been committed to.

Example: Performance measurements are collected from...
all ongoing projects. Insufficient performance is not punished but carefully analyzed. Mostly it is found that too many changes still appear for no reason. As a consequence, a strong focus is given towards change management and change review boards. A weekly project review is introduced and after a few weeks enhanced with daily Scrum meetings of development teams. Requirements changes passing the change review board must have their proper business case or are not accepted. Marketing and sales are unhappy – but the results prove valid. It is they who do not want to take ownership and be held accountable for changes. This reduces changes within the first three months dramatically. The first few projects have 20% schedule overrun (compared to the previous average of 45%) and two of them are even close to 10%. These two are further evaluated to identify best practices. As it turns out, the project managers demanded requirements reviews by product managers and testers before submitting requirements to change review boards. The quality of requirements substantially improved. This change is immediately pushed forward by senior management to all projects. As it turns out, testers are unhappy because they have not got the time scheduled for doing the reviews. Rather than demanding overtime, senior management asks the projects to budget 5% slots each week for reviews. These are two hours which are sufficient to review 1-3 requirements with sufficient depth.

A goal-oriented measurement approach ensures that process improvement is embedded in a closed feedback loop (see Figure 7). Business-driven improvement objectives are translated into annual targets or key performance indicators. Those are reflected in, and tracked with scorecards. A history database captures process, product and project information. It helps in tailoring processes and with setting specific process and project targets. It also facilitates setting control limits for statistical process control. The feedback loop is built upon measurements from processes that are analyzed against control limits and compared with targets.

Questions such as "are we doing better or worse?" will provide feedback from the currently running engineering projects where changes should be integrated into the workgroups responsible for making such changes happen. People should make realistic commitments and are later held accountable for achieving them. Visibility means trust. The figures are not "made up" for reports; they are a normal (self-) management tool. Not providing visibility or not delivering committed results is failure. Deviations that are out of a team’s or a project’s own control are flagged in a timely manner to allow resolution at the next highest level. Accountability also means setting realistic and measurable objectives. Objectives like "reduce errors" or "improve quality by 50%" are pointless. The objective should be clear and tangible, such as "reduce the number of late projects by 50% for this year compared to the previous year." These objectives must end up in a manager’s key performance indicators. Goals that are measured will be achieved!

Process measurement does not stop with simple indicators for efficiency or effectiveness. In fact, what we have just described is just the beginning of process measurement. Full quantitative process management uses statistical techniques to identify process anomalies, to eliminate them and to ensure the processes will deliver what is asked for by the business (see Figure 8).

Process measurement values represent process behaviors over time, for instance, defect detection rates in different test cycles. The business needs to determine specification limits of the processes, such as a maximum of known defects released to the field. Often these specification limits are asymmetric.

In our example of field defects, a lower specification limit could be impacted by time to market, thus indicating that the product must be of the right quality. Internal process behaviors determine control limits. Control limits are determined by the way the process is defined, disseminated, automated, and executed. They represent the "natural" behavior of the process if executed as specified. For instance if we have a life-cycle with six different quality gates and respective verification and validation steps, we might end up with a remaining defect level of around 5% (assuming we are effective in removing 60% of defects at each step.). Depending on the volatility of the process, these control limits are inside the specification limits and allow adjustments, if for instance a higher quality level is required.

Having 5% residual defects might be good enough if evaluated with a reliability prediction model based on operational usage schemes. It would also need to have a support and service organization in place that can provide the necessary assistance to customers as it might be demanded by service level agreements.

We call this natural behavior of the process the "voice of the process" (VOP). If this natural, built-in process behavior is consistently achieved and exceptions are resolved immediately to prevent any trend away from the control limits, we consider the process to be stable. It is good enough to achieve what we have designed it for. If the voice of the process is within the requirements specification as determined from business needs or the "voice of the customer" (VOC), the process is called capable.

For immature organizations, the control limits are mostly outside the specification limits. Those organizations are have no idea how to adapt processes so that they tend towards specification limits. In fact, these organizations do not even think in terms of control limits, they just hope each time a project is started that it reaches the requested target date or quality level. But process volatility yields continuous surprises. The process is not capable. This explains why the CMMI has in its name the word capability.

In quantitative process management we distinguish between the voice of the process (VOP) and the voice of the customer (VOC). Improvement objectives are the voice of the customer. Business needs and the customer base determine how good processes need to be. Capability baselines are used to identify the normal process behaviors (VOC) and to compare to the process needs (VOC). A small exam-
Experiences and Advances in Software Quality

Figure 7: Goal-oriented Measurement Ensures that Process Improvement is Embedded in a Closed Feedback Loop.

Figure 8: Improving Process Performance with Quantitative Process Management.
more specific VOC derived from the high-level VOC. They are sufficiently detailed to start improvement actions. They do not however answer how to improve the respective processes. This is your own insight and experience. Often you need external help for this step and to identify appropriate improvement actions.

Obviously, the E4–measurement process that we introduced before also applies here. The improvement process has the steps of setting improvement or performance objectives, measuring the process, analyzing the process behaviors and improving the process by first stabilizing it, then keeping it within its specification limits and finally continuously improving it. This goal-driven approach together with using statistical techniques to evaluate trends, behaviors and relationships, is formalized in techniques such as Six Sigma or CMMI [1][7].

5 Practical Measurement Guidelines

Measurement programs mostly fail because they are disconnected from actual business. Too often things are measured that do not matter at all. Alternatively there is a data cemetery with lots of unused data which is just reported because "we need metrics". Perhaps there is no clear improvement objective behind what is measured. The primary question with software measurement is not "What measurements should I use?" but rather "What do I need to improve?" It is not about having many numbers but rather about having access to exactly the information you need to understand, to manage, and to improve your business. This holds for both project and process measurement.

As a professional in today’s fast-paced and ever-changing business environment, you need to understand how to manage projects on the basis of measurements and forecasts. You need to know which measurements are important and how to use them effectively. Here are some success factors for your projects:

- Estimate project time and effort and set realistic deadlines.
- Check feasibility on the basis of given requirements, needs and past project performance (productivity, quality, schedule, and so on). Do not routinely over-commit.
- Manage your requirements and keep track of changes. Measure requirements changes and set thresholds of what is allowed.
- Know what the stakeholders judges as the value of the project. Track the earned value of your project.
- Understand what is behind delays and defects. Do not let delays accumulate. Remove root causes and do not simply treat symptoms.
- Be decisive and communicate with a fact-based approach. Avoid disputes with management, clients and users.
- Use measurements to keep commitments on track.
- Continuously improve by means of looking to your measurements and then taking specific objective-driven actions. Follow through until results are delivered.

As a practitioner encourage your management to take decisions on the basis of measurements. Give them the necessary information before and after the decision so that they can follow the effects and compare to goals. As a manager ensure that you decide based on facts and analysis. Always consider the fourth E-letter in the E4–measurement process, namely to execute. Make sure that your decisions move your business towards agreed objectives.

References


Further Reading

C. Ebert, R. Dumke. Software Measurement. Springer-Verlag, Heidelberg, New York, 2007. ISBN-13: 978-3540716488 (reference no. 1 of this article). Description: People who design and develop software like to call themselves software ‘engineers’. Yet few organizations have really institutionalized measurement of their products and processes. The book is right up-to-date in both fields and packed with practical advice which make it the desktop reference for software engineers when it comes to working with software measurements. It provides the necessary foundations for measurement and has many concrete industry case studies to help to transfer the results to your own environments. Additionally it offers a large number of "rules of thumb" and benchmarking data that can be used in projects and in process improvement and assessment activities. This article takes some content from that book.
Methods for Testing Web Service Compositions

José García-Fanjul, Marcos Palacios-Gutiérrez, Javier Tuya-González, and Claudio de la Riva-Alvarez

The deployment of software as a service has the objective, in the short or medium term, that these services will be invoked not just from one particular application, but also from other software or services. Consequently, using well-established and automated testing methods is essential to firstly assure the quality of the deployed services and also to facilitate regression testing. In this paper, we describe methods that have been recently proposed to test web service compositions, particularly focusing on the de-facto industrial standard BPEL.

Keywords: BPEL, Dynamic Binding, Monitoring, Software Testing, Web Service Compositions.

1 Introduction

Service Oriented Architectures (SOAs) have become, in recent years, a common standard for building and deploying software. In such architectures, software is divided into decoupled services which are composed to build complex applications. As with any new technology, SOAs have a set of advantages such as the low coupling between the services. This characteristic enables binding with third-party provided services or even establish dynamic binding policies. However, the adoption of this new technology has also raised concerns regarding the software development processes and, more precisely, software testing processes. Canfora and Di Penta [1] and Zhang and Zhang [2] identified a number of unresolved challenges in the application of traditional software testing technologies to services such as:

1. The need to remotely test web services, with its associated cost.
2. The impact that the limited information exposed about a web service has on the design of test cases.
3. The ability to dynamically search and invoke web services.

Since these challenges, among others, were identified, research groups have worked on the definition of testing technologies useful for Service Oriented Architectures. This is a state of the art article on the testing methods committed specifically to uncover faults in web service compositions, particularly focusing on the de-facto industrial standard BPEL.

The article is organized as follows: Section 2 outlines the background concepts regarding services, compositions of services and software testing. The following three sections are dedicated to monitoring methods (Section 3), methods to test dynamic binding compositions (Section 4) and functional testing (Section 5). A discussion is given in Section 6 regarding common issues in research about software testing for SOAs and lastly, in Section 7, conclusions are presented.

2 Background

2.1 Service Oriented Architectures

A good definition of web service is given in the web services glossary made public by the W3C consortium [3]:

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.
As it is said in the definition, most of the existing services today are web services and thus they are invoked using the HTTP (Hypertext Transfer Protocol) protocol. Furthermore, most of the services today have their interfaces described in terms of the WSDL (Web Services Description Language) standard and they exchange data in XML (eXtensible Markup Language) format. Nevertheless, and bearing in mind that there exists a great amount of deployed services which are not web services (as they can be invoked, for example, using a protocol other than HTTP) in this article we tend to use the term "service" and not the more restrictive "web service".

Service providers may publish their characteristics in brokers or registries that commonly use the UDDI (Universal Description, Discovery and Integration) standard. Such brokers may be queried by applications that need to use a certain functionality, and they will reply with a number of suitable services. Applications will then select a service, dynamically bind to it and invoke it. This software design is called Service Oriented Architecture (SOA).

A composition of services is distributed software that invokes an amount of services such that services may come from different providers. A composition of services can be executing different software each time it is executed, because it cannot be controlled if providers change the implementation of their services. Furthermore, in dynamic binding contexts, a composition of services may be executing different services each time it is executed.

Compositions of services are classified conceptually, as orchestrations or choreographies. Orchestrations require that there is one service which invokes all the other services in the composition. The de facto standard to implement orchestrations of web services is BPEL (Business Process Execution Language) [4]. Service choreographies do not need a central coordinator, and the coordination responsibility is shared among all the participating services.

2.2 Software Testing

Testing is one of the fundamental processes for quality control of software. One of the best definitions for software testing comes from the IEEE standard Std610.12-1990 [5]:

The process of analyzing a software item to detect the differences between existing and required conditions (that is, bugs) and to evaluate the features of the software items.

It is commonly understood that testing implies that the software under test should be executed (dynamic testing). However, the above definition includes any analysis made to a software item as a test, and therefore it also includes static checking techniques, in which software is not executed to find bugs.

The traditional approach to software testing required that most of the tests should be executed before the software is deployed in production environments. The point was that quality control processes should be performed before the product is released to customers. Nowadays, due to the emergence of Service Oriented Architectures, this approach is complemented with monitoring techniques - also called online testing [6]. These techniques are oriented towards continuous inspection of software behaviour in production environments, and are often complemented with self-adaptation strategies, in case misbehaviour is detected.

3 Monitoring

One of the main characteristics of Service Oriented Architectures is low coupling. Service providers may introduce changes in the behaviour of their services in a number of ways such as, for example, deploying a new version of the service, or modifying computational resources dedicated to the execution of services.

Thus, from the point of view of service clients, they must define strategies to find out, in runtime, whether there are misbehaviours in their service compositions due to changes in the behaviour of the composed services. As mentioned above, the testing techniques oriented towards continuous monitoring of software behaviour in production environments are monitoring techniques.

As with any process that must be executed in production settings, monitoring techniques should be able to detect faults by observing the execution of the composition of services with a minimum interference in the behaviour or workload of the composition. Monitoring techniques that can be implemented with no change in the behaviour of the composition or the services which are under test are often called passive monitoring.

In this kind of technique, it is common to deploy probes that inspect the messages (typically SOAP, - Simple Object Access Protocol -) that services participating in the composition interchange. A fault is therefore detected when the obtained messages do not comply with the specification of the service composition. For example, Raimondi et al. [7] propose deploying probes in service providers or clients. The probes check whether Service Level Agreements (SLAs) are met in terms of quality of service (QoS) properties. Their paper describes how to systematically obtain probes from SLAs specified in the SLAImag language.

A similar proposal [8] describes an architecture of probes that monitor functional and non-functional behaviours in service compositions. The difference between this and the former proposal is that the specification of the expected behaviour for the composition of services is made in an augmented version of Finite State Machines.

In some situations, passive monitoring techniques cannot be applied. Therefore, the composition or the services must be modified to instrument their code and obtain information about their behaviour when executed. These techniques are called active monitoring. A relevant proposal in active monitoring [9] describes how to deploy a tool to monitor the execution of BPEL processes. The tester defines rules that specify the expected behaviour of the composition, and these rules are inserted into the BPEL code as comments. The BPEL engine is complemented with a plugin that is able to interpret such rules and decide whether the observed behaviour conforms to the rules or not.
The papers described above use the probes to discover faults as soon as the faulty software is executed. This is the most common approach for monitoring compositions of services, but some faults cannot be discovered using such approaches. Offline monitoring techniques gather information about the behaviour of compositions of services and then offline process that data to establish whether there are faults.

The most representative example is the article by Rozinat and Van der Aalst [10]. In this article, they propose a method to determine, using Petri Net based tools, whether the executions of a composition of services conform to a BPEL specification of the composition.

4 Testing Dynamic Binding Compositions

A key feature in SOA architectures is their capability to change the services that participate in a composition. Service clients may decide, in runtime, which service will be invoked among the suitable ones. This characteristic is called “dynamic binding” and it is implemented using brokers (or registries) of services. Service providers publish the features of their services in the broker, and service clients query the broker and decide which of the available services will be invoked. Figure 1 illustrates this process.

Dynamic binding raises certain challenges from the point of view of software testing. For example, brokers may wish to decide whether the services they publish hold certain QoS properties. Test suites may then be designed and successfully executed on candidate services as a precondition to be referenced by the broker [12]. The broker may also periodically monitor to ensure that services are continuously complying with the defined requirements in terms of QoS [13].

Both of the above mentioned papers propose to extend the UDDI standard, as UDDI defines the functionality of the broker as a passive registry. UDDI brokers just store the necessary information to realize the dynamic binding, and

![Figure 1: Service Brokering - adapted from Papazoglou et al. [11].](image)

not semantic data related to the behaviour of the referenced services.

Other proposals take advantage of the dynamic nature of service invocations so that when a fault is detected in a monitored composition of web services, the invoked service may be changed for another one that successfully executes a test suite, and so it is assumed that it complies with the specification. For example, Moser et al. [14] have designed a tool to replace, in runtime, services which are composed in a BPEL process. The replacement is performed when the monitoring framework identifies that QoS criteria do not hold. In another approach a framework is described to decide which of the available services will be selected to be bound in a composition, using genetic algorithms to estimate the best solution in terms of QoS [15].

5 Functional Testing

There is an immense diversity of situations that may be reproduced as a test with the intention of uncovering faults in software functionality. In practice and in most of the projects, the amount of different test situations may be considered infinite. Thus, functional test designers must select test cases that, when executed, have a high probability of finding faults and, also, can be fully designed and executed within the available resources. Research in functional testing and test case generation has, therefore, the objective of finding good test suites for testing a given software, that optimize the above described criteria.

Commonly, approaches to generate functional test cases for software rely on techniques that make a specification of the expected behaviour of the software under test, and test cases are derived from the specification applying certain algorithms. Specifically, in the field of functional testing for compositions of services, model checking (a formal method) has been used in several approaches [15][16]. In the first of these papers, a model is obtained from the specification of the composition in BPEL, and an adequacy criterion (transition coverage) is defined to systematically obtain test cases applying a model checker. In the latter, the composition is specified in OWL-S and no criteria are defined to obtain the test cases: they are obtained from properties which are specified by hand.

Other approaches use different formalisms to obtain the test cases. Mei et al. [17] describes a data-flow technique to obtain test cases for BPEL processes relying on term-rewriting tools. Another paper prescribes a control-flow method and expounds how to generate tests, also for BPEL processes, from a model of the flow of BPEL activities and using a constraint solver [18]. Another technique, Petri Nets, that is commonly used to generate test cases for software testing, is applied to test compositions of web services in which the desired behaviour is extracted from contracts [19].

The majority of the research conducted on functional testing for SOAs is dedicated to the unit testing of a service or, at most, a service that is executed in the scope of a composition of services.
the rest of the composition, other services must be deployed to work as stubs. The design and implementation of such stub services may be automatically feasible if the behaviour of the composition is specified [20]. Other frameworks and tools for designing and executing unit tests for services are described in the papers of Li et al. [21] and Mayer and Lübbe [22]. In both it is possible to unit test services that participate in a composition that is specified using BPEL. The work of Mayer and Lübbe was later implemented into a tool called BPELUnit [23].

6 Discussion

For Service Oriented Architectures, there exist standards that are commonly accepted to describe the interfaces of the services. The growing enthusiasm for these standards in industrial settings is one of the key advantages of these architectures. But up to now no consensus has been reached with respect to standards to describe service behaviour, which is not a new problem [24].

If the papers that are referenced in this article are reviewed, most of them use one technique or another to specify behaviour of services or compositions of services, such as BPEL, OWL-S or SLAng just to mention three. Some of the researchers even propose new ad-hoc techniques to specify behaviour. In the field of testing techniques, the lack of commonly accepted standards to specify service behaviour leads to difficulties in establishing systematic strategies for the execution of tests, adequacy criteria for selecting test cases or the definition of oracles to set the expected behaviour of test cases. One of the fields in which the lack of these standards is more visible is in the testing of dynamic binding compositions of services. The de facto standard for broker architecture (UDDI) is systematically complemented in research papers, with semantic information that allows registering or querying services applying different criteria depending on the service behaviour.

New standards must be adopted to define functional and non functional behaviour of services and compositions of services. Nevertheless, it must be noted that the latter (non functional) characteristics are the ones receiving more attention in research works in the fields of monitoring and dynamic binding.

This trend may be influenced by two concerns that final users commonly state about SOAs. First of all, the need to show that SOA solutions, like any other software, fulfill a number of QoS requirements to warrant that it is usable, before considering whether there are problems in the functional requirements. On the other hand, several final users feel that functional faults are easier to detect and correct than QoS faults, so more resources are employed in availability or performance testing rather than in functional testing.

It should also be stressed that many of the papers published in the field of testing SOAs are not validated against industrial-size examples. Many of them are just exemplified with simple compositions of services that are extracted from standards or designed ad-hoc for the research. It is of utmost importance to define standard examples that can be used to validate the researches, and compare the results obtained on the application of different testing techniques. Industrial sized examples must also be published to evaluate the scalability of the research approaches.

7 Conclusions

In recent years, researchers have worked on different techniques to test compositions of services. One of the fields in which there have been a greater number of proposals is the monitoring of SOAs, an especially relevant matter, bearing in mind the low-coupled nature of services.

Regarding the testing of dynamic binding compositions, there are just a few research proposals and much work needs to be done in the future, even at the conceptual level. A great amount of work has been published about functional testing of SOAs, using techniques that were productive in the past for other kinds of software, but there is also a need to focus on the quirks of compositions of services.

Advances in the testing of SOA software could be made if there were commonly adopted standards for the specification of the behaviour of services. The only standard that has been clearly adopted at the industrial level has been BPEL. This language allows specification of the functional behaviour of compositions of web services but there is a growing need to define and adopt standards for the definition of functional and non functional properties for individual services.

Acknowledgements

This work is supported by the Ministry of Science and Innovation (Spain) under the National Program for Research, Development and Innovation, projects Test4SOA (TIN2007-67843-C06-01) and RepRIS (TIN2007-30391-E).

References

Experiences and Advances in Software Quality

Atlanta (Georgia – USA); pp. 170-180.


A Quality Evaluation Model for Web2.0 e-Learning Systems

Stephanos Mavromoustakos and Katerina Papanikolaou

Web2.0 is used in e-Learning to transfer the qualities of social networking to the virtual classroom. The continuous evaluation of the Web2.0 e-Learning systems requires the use of quality evaluation models in a continuously evolving environment. This paper proposes a quality evaluation model for Web2.0 e-Learning systems called EEQM which focuses on three main components: a) software quality factors, b) pedagogical requirements and c) Web2.0 elements. The EEQM model involves the combination of three methods for estimating the quality of e-learning systems: a) Frequency, b) Median, and c) Total Quality.

Keywords: e-Learning, Evaluation Model, Software Quality, Web Quality, Web 2.0.

1 Introduction

Software quality evaluation is essential to the production of software products that are reliable, efficient, understandable and acceptable to their stakeholders [1]. In developing Web applications it is also necessary to build and utilize sound evaluation models and tools so as to facilitate and ensure the continuous quality of those applications. In recent years, several style guides, lists of guidelines, and design principles have emerged to assist developers in the development process. While these guidelines and techniques are helpful in the Web design process, they do not necessarily constitute an evaluation method in themselves.

Several researchers have proposed various recommendations or evaluation techniques for Web applications [2][3][4][5][6][7]. However, quality evaluation of e-learning applications is a new and often neglected issue. With the advent of Web 2.0 in almost all areas of Web applications as well as in e-learning applications, a sound and practical evaluation model is necessary.

The E-Learning Evaluation Quality Model (EEQM) proposed in this paper is based on a three axis approach: a) software quality factors, b) pedagogical requirements, and c) Web2.0 elements. The quality approach is based on internal and external evaluation criteria. Internal refers to organization Web experts, and external to a group of users (learners). The EEQM model combines three methods for estimating the quality of e-learning systems: a) Frequency, b) Median, and c) Total Quality.

The rest of the paper is organized as follows: Section 2 describes the three main quality criteria, Section 3 explains the EEQM model, and Section 4 describes step-by-step the proposed evaluation process. Finally, Section 5 draws the concluding remarks and suggests some steps for future work.

2 Quality Criteria

This paper aims at providing a systematic method for evaluating Web 2.0 e-learning systems, utilizing three main components: a) software quality factors, b) pedagogical requirements, and c) Web2.0 elements.

2.1 Software Quality Factors

The software quality factors component proposed in this quality evaluation model takes into account ISO 9126 and its decomposition into several quality factors, as proposed in the context of Web engineering and W3C standards and recommendations.

ISO 9136

ISO 9136 refers to:
Usability: Issues such as understandability, learnability, friendliness, operability, playfulness, and ethics are vital design factors that Web engineers cannot afford to miss. The system must be implemented in such a way as to allow for easy understanding of its functioning and behaviour, even by non-expert Internet users. User interface aesthetics, consistency, and ease of use are attributes of easy to learn systems with a rapid learning curve. By keeping a user profile and taking into consideration human emotions, e-learning systems can deliver related messages to the user, whether a welcome message or an order confirmation note, thus enhancing the friendliness of the system. Playfulness is a feature that should be examined to see whether the application requires this characteristic and, if so, to what extent. E-learning systems must reflect useful knowledge by looking at human interactions and decisions.

Functionality: The system must include all the necessary features to accomplish the required task(s). Accuracy, suitability, compliance, interoperability, and security are issues that must be investigated when designing an e-learning system, to ensure that the system will perform as it is expected to. An e-learning application must have search and retrieve capabilities, navigation and browsing features, and application domain-related features.

System Reliability: Producing a reliable system involves understanding issues such as fault tolerance, crash frequency, recoverability and maturity. The system must maintain a specified level of performance in the event of software faults with the minimum number of crashes possible. It must also have the ability to re-establish its level of performance. A system must consistently produce the same results, and meet or even exceed users’ expectations. The e-learning application must have correct link recognition, user input validation, and recovery mechanisms.

Efficiency: An e-learning system’s goal is usually to increase productivity, decrease costs, or a combination of both. Users expect the system to run in an efficient manner in order to support their goals. System’s response-time performance, as well as page and graphics generation speed, must be high enough to satisfy user demands. Fast access to information must also be examined throughout the system to ensure that users’ requirements are continuously met on the one hand, and that the system remains competitive and useful on the other.

Maintainability: Some crucial features related to maintaining an e-learning application is its analysability, changeability, stability, and testability. The primary target here is to collect data that will assist designers to conceive the overall system in its best architectural and modular form, from a future maintenance point of view.

Given the rapid technological changes we are seeing, especially in the area of Web engineering, as well as the rigorous demand from users for continuous Web site updates, easy system modification and enhancement, both in content and in the way this content is presented, are also success factors for the development and improvement of an e-learning system.

W3C
The World Wide Web Consortium (W3C) aims to provide Web standards for Web interoperability. In addition, W3C provides a Markup Validator, a tool that verifies whether a document actually follows the rules of the markup language used. This is the process of checking a Web document against the grammar (typically a DTD) it is using.

W3C also provides guidelines for a quality Web application on:
- Title.
- Avoiding the "click here".
- Using redirects.
- Using headers.
- Using the alt attribute.
- Using a doctype.
- Using the <link> element.
- Using CSS rather font.
- Using international date format.
- Using PNG rather GIF.
- Using headings.
- Choosing and managing URIs.
- Using quality content.
- Using class with semantics in mind.
- Using unordered lists.

2.2 Pedagogical Requirements
Transferring the dynamic nature of learning to the new e-learning environment, maintaining student individuality and differentiation according to personal preferences and abilities, as well as motivating and inspiring students, are all key factors for the acceptance of the new learning environment [8][9].

The key factors are identified as follows:
Identification of learners’ needs: The e-learning environment should be shaped according to predefined learners’ needs and the course required pedagogical outcome.
Structuring of the pedagogical material: The pedagogical material should be constructed in a way that facilitates the successful transfer of the required knowledge.
Enhancement of the e-learning environment: The e-learning environment can be used either as complimentary or in parallel to the real classroom environment. In either case the e-learning environment should adhere to the basic mechanisms and functions of the real environment. In the case of pure distance learning this enhancement is even more imperative.

Motivation for student participation: Transferring to a virtual environment is not always straightforward or easy. Students are not always willing to use a virtual environment for a number of reasons, such as the difficulty of the e-learning tool, the non-intuitive nature of the environment, problems of low interactivity, etc. The e-learning environment should have mechanisms to allow questions/answers sessions similar to those in a traditional classroom. The e-learning environment should be able to offer the students a basic problem solving mechanism. Mechanisms such as online tutorials, contact with the instructor, reference to use-
ful resources, and even access to a technical helpdesk would offer students support and help.

The establishment of collaborative mechanisms among students: In the virtual environment the student can be easily isolated and separated from the rest of the class. This is usually avoided in the real classroom and should be avoided in the virtual classroom too, by organizing and operating on a collaborative basis so that students can interact and communicate.

The use of relevant tools and components for the support of any specific solution: Depending on the target student audience and the required learning outcome, the appropriate tools should be implemented and differentiated accordingly. Vocational training requires different solutions from educational training and undergraduate training has different pedagogical targets from undergraduate training. Tools and components can be utilized to enhance the e-Learning environment more efficiently.

The right mix of the learning processes implemented: The most important learning processes are identified as follows: analysis, synthesis, reasoning, judging, problem solving, collaboration, simulation, evaluation, presentation and relation. These processes should be used dynamically to construct the learning scenario for each course and student.

2.3 Web 2.0 Elements

One of key factors affecting e-Learning effectiveness was identified to be the lack of interactivity [10][11][12]. The isolation of the learner behind a screen leads to falling motivation, loss of interest, and failure. In parallel, the same need for user interactivity has its effects on the Internet and its use. Internet users realized that the asymmetrical flow of information made most of them content consumers rather than content providers. But the Internet can inherently provide access to users both as content consumers and content creators.

The realization of this fact led to the establishment of Web2.0. Web2.0 is harnessing the Web in a more interactive and collaborative manner, emphasizing peers' social interaction and collective intelligence, and presents new opportunities for leveraging the web and engaging its users more effectively. In the last three years, Web2.0, ignited by successful Web2.0-based social applications such as wikis and blogs and application specific software such as my MySpace, Flickr and YouTube, has been forging new applications that were previously unimaginable.

In this section, we present the basic applications enabling social networking and used in e-Learning interactivity.

Weblogs

The Web offered the perfect medium for immediate press releases and fast dissemination of news and information. The publishing on the Web of an individual’s diary, with the thoughts, views, comments and positions created the revolution of Web-Logging or Blogging.

The term was initially coined by Jorn Barger in 1997 and in its simplest form is a website with data entries, presented in reverse chronological order [13]. This is the outcome of a common need for the sharing and expression of thoughts, criticism and experiences by individuals and was, from its outset, one of the strongest tools of the Internet. A Blogger is the owner of a Blog and contributing to a Blog is called blogging. Each Weblog is part of the Blog-o-sphere. The number of existing blogs is rapidly growing and there seems to be no end in the near future. The new-found "democracy" has many supporters but there are concerns regarding the extent of freedom of speech. While the concept of an on-line diary is far from new, the popularity of Blogs is increasing rapidly.

There are two main reasons for their success, as identified by other researchers [14]:

1. Personalization: A Blog is personal and expresses the authors’ views and ideas but others can contribute too. A Blog provides directness of communication and a forum for discussion and the exchange of ideas.

2. Usability: The crucial factor for the success of Web2.0 applications is ease of use. They are not aimed at people with technical computer programming skills. Everyone is able to contribute to the WWW and become a content creator by clicking on his/her weblog, logging on, and writing with the help of a WYSIWYG-Editor.

The amount of information trafficking in Blogs can be enormous, and to cope with this a personal overflow RSS (Really Simple Syndication) technology is used. With the help of XML structure, an RSS-Reader can provide feeds from subscribed Blogs or other applications. The big advantage is that new information can be read without opening a site. Furthermore, the possibility of using Aggregators and Search functions help to make the information consumption more efficient.

The popularity of blogs has raised concerns regarding the legal and other ramifications of releasing confidential information, use of language etc.

Wikis

A wiki is a simple yet powerful Web-based collaborative-authoring (or content-management) system for creating and editing content. It lets anyone add a new article or revise an existing article through a Web browser. Users can also track changes made to an article. The term wiki is derived from the Hawaiian word wikitiki, which means fast or quick. The user-generated online encyclopaedia Wikipedia (http://en.wikipedia.org) is a wiki.

Wiki features include:

- **A wiki markup language.** "Wikitext" provides a shorthand way of formatting text and linking external documents and contents.

- **Simple site structure and navigation.** Contributors can create new pages and easily link one page to another. Because a blog site's hierarchy and structure is flat, navigation is simple.

- **Simple templating.** When a page of wikitext is requested, wiki software converts the wiki markup to HTML and creates links between pages, then wraps this converted
content in a template to provide a consistent look to all pages in the wiki.

- Support for multiple users. Hyperlinks to pages within the wiki are created automatically. Wiki software makes links based on the page’s title, so the author does not need to use, remember, or type long URLs to link one page to another within a wiki.

- Simple workflow. You can write or edit and publish without editorial oversight or approval. Content in a wiki is managed through change monitoring and the wiki’s ability to roll back to a previous version and prevent spam. You can also control user access and privileges, if required.

- A built-in search feature. You can search for specific information or topic within a wiki using associated keywords.

Wikis facilitate collaborative work and this is their main difference from Blogs. Due to this collaborative ability, wikis can significantly enhance the learning environment.

Mashups

A Web mashup is a Web page or Web site that combines information and services from multiple sources on the Web. Web mashups can combine information and/or complementary functionality from multiple Web sites or Web applications. A Web mashup server lets you connect, collect, and mash up anything on the Web, including data on some backend systems. There are seven major categories: mapping, search, mobile, messaging, sports, shopping, and movies. More than 40 percent of mashups are mapping mashups. [15] Several other new-breed Web applications similarly integrate multiple services under a rich user interface.

Typical applications are HousingMaps <http://www.housingmaps.com>, that display sales and rental information from a classified ads Web site into Google Maps. Users can view the map enhanced with information on what property is available for rent or sale in the area. Another example is Fishing Solutions <http://www.fishingsolutions.com.au> that uses Google Maps and information from anglers to help users find fish.

It is easier and quicker to create a mashup than to code an application from scratch in a traditional way. This capability is one of Web2.0’s most important and valuable features.

Podcast

A Podcast is defined in Wikipedia as follows: "A Podcast is a multimedia file that is distributed by subscription (paid or unpaid) over the Internet using syndication feeds, for playback on mobile devices and personal computers". In the beginning, multimedia files were the same audio files (.mp3). Nowadays video files are also distributed via Podcasts.

Similar to Weblogs, the technology behind them is rather simple. Their popularity is due to the ease with which users can produce their own Podcast, with the help of RSS, the growing bandwidth of Internet connections (which make bigger downloads possible), and the availability of Internet-enabled mobile devices. Some examples that describe the use of Podcast in Education can be found in [16][17].

It seems that this technology is gaining in popularity. The need to include social and collaborative elements in e-learning academic environments has already been identified [14].

3 The EEQM Model

The EEQM process includes a three axis approach to quality based on software quality factors, pedagogical requirements and Web 2.0 elements. The Software Quality axis aims to evaluate all Web elements related to the quality of the e-learning system. The evaluation process is performed internally by the organization (e.g. academic institution). Internal in this paper refers to the sub-process of the first axis. It is defined as the evaluation performed by the organization’s experts (managerial and technical) and

![Figure 1: The EEQM Framework.](https://example.com/eeqm.png)
views the Web application as a business tool and as a system. The management defines and verifies whether the Web application meets the organizational/business goals and the specific targets of the organization. The technical personnel evaluate the application characteristics based on general software quality standards and on certain Web quality factors as described in the previous section.

The Pedagogical axis aims to evaluate all the requirements necessary for an e-learning system. The evaluation process is performed internally and externally. External in this paper refers to the evaluation performed by a group of users. A group of learners evaluate the Web system based on their own requirements and expectations.

The Web 2.0 axis aims to evaluate social activities and, in particular, collaboration among learners. The evaluation process is performed only externally by a group of learners. Figure 1 shows the EEQM framework that was devised illustrating the three axes of the evaluation model and the boundaries between the internal and external environments.

A sample users’ questionnaire is developed for evaluating the quality of the Web domain under examination by measuring user satisfaction and the importance of each criterion (see appendix A). The questionnaire was designed with a focus on quality factors, as described earlier. For example, certain questions are related to usability, others to functionality, and so on. In addition, Web quality factors as proposed by researchers [18][19][20].

Table 1 is used for evaluating the various components that affect the quality of the Web application. The users fill out the questionnaire and the organization technical experts define the important quality factors from Table 1.

Both types of evaluators, technical experts and users, indicate their satisfaction by assigning numerical values ranging from 1 to 5 as follows:
1. Very Unsatisfactory.
2. Unsatisfactory.
3. Good.
4. Very Good.
5. Excellent.

The evaluators also assign numerical weights ranging from 1 to 5 to each requirement so as to distinguish the different levels of importance. The higher the number, the more important the requirement:
1. Least Important.
3. Important.
4. Very Important.
5. Most Important.

Since the weights are only estimated, only single digits of precision are used. The same also applies to satisfaction levels.

The EEQM model involves the combination of three methods for estimating the quality of e-learning systems in order to look at the Web application from different perspectives:

Method 1. Frequency: The frequency of users’ answers based on their satisfaction and importance for each question of the questionnaire will indicate users’ frequent impression of the Web application’s quality. Specifically, we can observe the number of users that are satisfied or not with a given feature together with their importance preference on a scale from 1 to 5. If the frequency of users’ satisfaction value is equal or more than 4 then we may consider that the Web application is ‘Satisfactory’, otherwise certain features may need further enhancement.

Method 2. Median: Median indicates the middle value of a set of data. The median is less sensitive to extreme scores than the mean and this makes it a better measure for highly skewed distributions. The median is the \( M^{th} \) value in a distribution \( L \) of length \( n \), where \( n \) is the number of values in a sorted set of data. We calculate the median for each question by using the formula:

\[
M = \frac{n}{2}, \text{ if } n \text{ is even}
\]

\[
M = \frac{(n+1)}{2}, \text{ if } n \text{ is odd}
\]

The calculation of median can provide more details about the quality of a specific feature/component. Features with values below the median need improving and priority should be given to those where the difference between the importance median and the satisfaction median is more than 2 points (e.g. Importance is 4 and Satisfaction is 2).

Method 3. Total Quality: To find the total quality of the Web application we first estimate the median for each feature/component for the internal evaluation (axis 1) as described earlier, for each question asked for the external evaluation (axis 3) and for each feature/component and question for both the internal and external evaluation (axis 2). Then to calculate the total quality for the external evaluation we use the weight to give us the number of times the satisfaction median will appear in our sorted set of data. The median of this set will be the total quality for the external evaluation. For example, if the satisfaction median for question 1 is 3 and importance (weight) is 4 then number 3 will appear 4 times in our set. For the internal evaluation we also add weights of importance for each expert. For example, a Webmaster may have a weight of 5 while the system manager has a weight of 4, a decision that has to be agreed between them. Therefore, for the internal evaluation we multiply each expert’s weight with the attribute’s weight and then we follow the external evaluation procedure. Based on our example, if the satisfaction of expert 1 for attribute 1 of the internal evaluation is 4 and its importance is 3, then the number 4 will appear in our set 15 times \((W_1 \times W_2)\). The same process will be performed for expert 2, and again for axis 2.

4 7-Step Evaluation Process

The EEQM model follows a step-by-step process to evaluate any given e-learning application. In particular the
Table 1: Quality Factors of Web Applications.

<table>
<thead>
<tr>
<th>QUALITY FACTORS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. FUNCTIONALITY</td>
</tr>
<tr>
<td>1.1 Accuracy</td>
</tr>
<tr>
<td>1.2 Suitability</td>
</tr>
<tr>
<td>1.2.1 Products taxonomy</td>
</tr>
<tr>
<td>1.2.2 Services taxonomy suitability</td>
</tr>
<tr>
<td>1.3 Interoperability</td>
</tr>
<tr>
<td>1.3.1 Browser independence</td>
</tr>
<tr>
<td>1.3.2 Hardware independence</td>
</tr>
<tr>
<td>1.4 Compliance</td>
</tr>
<tr>
<td>1.5 Security</td>
</tr>
<tr>
<td>1.5.1 Authentication</td>
</tr>
<tr>
<td>1.5.2 Payment transaction</td>
</tr>
<tr>
<td>1.5.3 Vulnerability</td>
</tr>
<tr>
<td>1.5.4 Confidentiality</td>
</tr>
<tr>
<td>1.6 Searching</td>
</tr>
<tr>
<td>1.6.1 Simple searching</td>
</tr>
<tr>
<td>1.6.2 Advanced searching</td>
</tr>
<tr>
<td>1.6.3 Sitemap</td>
</tr>
<tr>
<td>1.6.4 Search Feedback</td>
</tr>
<tr>
<td>1.7 Navigability</td>
</tr>
<tr>
<td>1.7.1 Links</td>
</tr>
<tr>
<td>1.7.2 Scrolling</td>
</tr>
<tr>
<td>1.7.3 Navigational prediction</td>
</tr>
<tr>
<td>1.7.4 Indicator of path</td>
</tr>
<tr>
<td>1.7.5 Shortcuts availability</td>
</tr>
<tr>
<td>1.7.6 Navigation error feedback</td>
</tr>
<tr>
<td>1.7.7 Mobile devices accessibility</td>
</tr>
<tr>
<td>1.7.8 Alternative access paths</td>
</tr>
<tr>
<td>1.8 Multilingualism</td>
</tr>
<tr>
<td>1.9 Printing mechanisms</td>
</tr>
<tr>
<td>2. EFFICIENCY</td>
</tr>
<tr>
<td>2.1 Time behaviour</td>
</tr>
<tr>
<td>2.2 Use of dynamic pages</td>
</tr>
<tr>
<td>2.3 Non-frames support</td>
</tr>
<tr>
<td>2.4 Text-only support</td>
</tr>
<tr>
<td>2.5 Performance</td>
</tr>
</tbody>
</table>

Table 1: Quality Factors of Web Applications.

EEQM model involves the following steps:
1. **Select the internal evaluators**: Identify the expert evaluators for the internal evaluation and the users for the external evaluation. The internal evaluators may include the project manager, the Webmaster, Web designers, a Quality Assurance Manager (QAM), etc. Usually, this is a small team of three to five persons. One of them leads the evaluation process. Preferably the QAM or in his/her absence the Webmaster. These evaluators are gathered together to perform the evaluation.
2. **Select the external evaluators**: Decide on the number of users and the way the external evaluation process will proceed. Users may evaluate the Web application either by filling in an online questionnaire or by taking part in an
evaluation focus group.

3. Establish the set of features to be evaluated and select evaluation criteria: The EEQM model can be utilized for the evaluation of the e-learning application as a complete system or for a specific group of features of the application depending on the organization’s requirements. Select from Table 1 those attributes from each quality factor that are more appropriate according to the type of Web application under study. Note that Table 1 may be enhanced by additional criteria if there are any other special characteristics of the application under evaluation.

4. Develop user questionnaire: Prepare a questionnaire for the users to evaluate the Web application focusing on revealing critical issues affecting the quality of the system. For the purpose of evaluation we have used a questionnaire as provided by Mavromoustakos and Androu [18].

5. Perform the evaluation: This stage consists of three parts. First, the evaluation is performed by the organization based on its requirements and following the aforementioned ISO quality characteristics and Web standards. The second part includes the evaluation by both the organization experts and a group of users to determine the quality of the system in reference to pedagogical requirements. Finally, the third part includes the evaluation by a group of users to determine user satisfaction, by indicating the quality of the e-learning application from their own point of view. Evaluators complete their questionnaires by selecting their level of satisfaction and by differentiating the high-importance level criteria from the low-importance ones.

6. Conclude the evaluation: After the questionnaires are completed, collect the data, analyse the questionnaires and calculate the results. The proposed formulas are used to indicate the mean, frequency and median of the Web application quality. Present the quality estimate for each feature and for the application as a whole. In addition, indicate missing features that need to be developed or existing features that need to be improved to achieve the desired level of quality.

7. Update the model: While the architecture of the EEQM model (the three-axes and the importance-based criteria component) should remain intact, the model can be adapted to new technological changes, as well as to new user requirements. Therefore, the model may continually be updated in terms of quality attributes in order to accommodate these changes.

6 Conclusions

In this paper we have addressed the issue of quality in e-learning applications and have proposed a three-axis quality evaluation model for Web2.0 e-Learning systems called E-learning Evaluation Quality Model (EEQM). These are: a) software quality factors, b) pedagogical requirements and c) Web2.0 elements.

The software quality factors of functionality, efficiency, usability, reliability and maintainability and their sub-components are studied in the evaluation of an e-learning application, enhanced by importance-based criteria.

The key pedagogical factors for enhancing the learning process includes the identification of learners’ needs, structuring of the pedagogical material, enhancement of the e-learning environment, encouragement of student participation, establishment of collaborative mechanisms among students, use of the relevant tools and components in support of any specific solution, and the right mix of the various learning processes implemented.

The EEQM model involves the combination of three methods for estimating the quality of e-learning systems: a) Frequency, b) Median and c) Total Quality. Finally, the model is represented in a practical stepwise process.

Further research steps will include the enhancement of the evaluation criteria in response to new user requirements and technologies emerging during the research. Finally, it would be interesting to look into the degree to which the EEQM model is applicable to mobile e-learning applications and to identify those parts of our approach that need to be modified and/or enhanced so as to facilitate the evaluation of this special group of software systems.

References


It seems very difficult to establish an objective criterion enabling us to distinguish thought from a mechanical imitation of it. The critical point is the definition of thought. In a 1950 paper Turing replaced the question, "Can machines think?" by a circumstantial problem based on a behavioural, dialogic procedure: if an interrogator is not able to distinguish the verbal behaviour of a computer from that of a human being, then it can be concluded that the computer can think. The so-called Turing test has been raising many debates for the vagueness of the original formulation and in connection with the advances achieved by artificial intelligence in the last decades.

"The method of Science is the method of all mundane knowledge; it is that of limitation or actual ignorance. Placed in face of the great uncontained unity of Nature we can only deal with it in thought by selecting certain details and isolating those (either wilfully or unconsciously) from the rest."

Edward Carpenter, Civilisation, its Cause and Cure

"Only the human brain can give a meaning to the blind ability of computers to create truth."

Karl Popper

Keywords: Artificial Intelligence, Body-Mind Problem, Gödel Theorems, Imitation Game, Loebner Prize, Turing Machine, Verbal Language.

1 Introduction

In 1950 the great British mathematician and logician Alan Mathison Turing (London 1912 - Manchester 1954) published a paper titled Computing Machinery and Intelligence in which he described a Gedankenexperiment, or thought experiment, aiming at determining whether a machine (a computer) can think. The experiment, that appeared on the journal Mind, has been known since as the Turing test and is inspired by a game, the so-called "imitation game", in which an interrogator tries to tell a man from a woman basing the decision only on their answers to a series of questions. Turing suggests to replace the woman with a machine...
Turing’s Biography in a Nutshell

Alan Mathison Turing was born in London, England, on June 23, 1912. Both his parents had upper middle class origins, and his father continued that tradition as an administrator in the Indian civil service. With the aid of a manual, little Alan learned to read by himself in three weeks. Turing was sent to private boarding schools. After some early problems with social adjustment, he distinguished himself in mathematics and science. Turing’s exceptional mathematical abilities were first recognized in his college years (1931-1936) at King’s College of Cambridge University. His most important mathematical work, *On Computable Numbers, with an Application to the Entscheidungsproblem*, was written in Cambridge in 1936. In this paper Turing introduced the concept of a theoretical machine (today known as the “Turing machine”) which embodies the concept of the stored program computer. In 1936 Turing was awarded a fellowship to visit Princeton University where he wrote a doctoral dissertation. During the Second World War he joined the Government Code and Cypher School in Bletchley Park, located between Oxford and London, where Turing played an important role in the design of equipment and development of techniques to break the German cipher system *Enigma*. He was close to the cryptanalyst group that designed Colossus, the first electronic digital computer with stored program, although he did not participate directly in its construction. Work at Bletchley provided Turing with valuable experience in electronics and with special-purpose calculating equipment which served him well after the war. In 1945 he moved to the National Physical Laboratory in Teddington, England, to assume responsibilities for designing an electronic computer, the ACE (*Automatic Computing Engine*). An ACE prototype served many important functions, including aircraft design, for several years. Towards the end of the war he was awarded the OBE (*Order of the British Empire*) for the brilliant services rendered in favour of Great Britain. In 1951 Turing became a Fellow of the Royal Society. Besides the articles referred to in these pages, Turing wrote two other important reports on the same subject: *Intelligent Machinery* (1948) and *Digital Computers Applied to Games* (1953), thus becoming a pioneer in artificial intelligence, the more so as he began investigating the relations between machines and the neurology and physiology of living beings. Actually from 1952 to his death in 1954, he devoted his genius to the mathematical formalization of biology, in particular morphogenesis, among other things exploring the connection between vegetal forms and Fibonacci numbers. In 1952 he was arrested for public practice of homosexuality, then a criminal offense in Britain. The courts mandated a dreadful penalty, chemical castration, that rendered him impotent. The depression that ensued was probably the reason why Turing committed suicide in his home in Manchester, eating an apple poisoned with potassium cyanide. Although contested by his mother, the hypothesis of suicide is almost sure. In her book *Zeroes and Ones*, Sadie Plant suggests that Apple Inc. chose the bitten apple as its logo in honor of Alan Turing, but the company has neither confirmed nor denied the conjecture.

and to assign the interrogator the task of discriminating between the man and the computer. The modified version of the game allows Turing to tackle the problem, *Can machines think?* avoiding the difficulties connected with the definitions of the terms "machine" and "think", whose meaning should be stated precisely from the outset.

The usefulness of this experiment lies not as much in the solution it provides to the original problem (it is not granted that it provides a solution) as in the possibility it offers to analyze such concepts as *mind, thought, intelligence*: in particular the concept of intelligence, that in those years was becoming the meeting ground of such disciplines as logic, mathematics, psychology and philosophy, that were in various ways concerned with man, or better with his cognitive capacities, considered as the most characteristic human abilities. It seemed important to possess a criterion if not to *define* intelligence at least to recognize its *presence* both in biological organisms and in artifacts. It should be emphasized that two years earlier, in 1948, Norbert Wiener had founded *cybernetics*, an ingenious discipline that aimed at studying "control and communication in the animal and

Figure 1: Alan Turing.
the machine" under a unified perspective. After a few years, in 1956, mathematician John McCarthy organized a summer school in Dartmouth College, at Hanover, New Hampshire, during which a tiny group of scholars, among whom Claude Shannon and Marvin Minsky, representing different but related fields of study, chose the name "artificial intelligence" (AI) to indicate a family of rather diverse research areas whose common objective was to produce simulations or emulations of intelligent behaviours in computers.

In the following sections we describe the Turing test and examine it critically to disclose its merits and its limitations. Although the discussion is self-sufficient, constant reference will be made to the original paper by Turing, that is still of great interest and to which we refer the reader (see Figure 1).

2 The Imitation Game

The imitation game consists in a linguistic exchange between three persons: a man (A), a woman (B) and an interrogator (C) who can be of either sex. C stays in a room apart from the other two and can ask A and B any kind of questions. On the only basis of their answers, C has to determine whether A is the man and B the woman or viceversa. Both A and B try to be identified as the woman, hence B endeavours to help the interrogator and A does his best to cheat C giving such answers as he thinks a woman would give. C has at his or her disposal only the answers given by A and B to the same series of questions. Their decisions can be set up.

Turing imagined to modify the imitation game, replacing A by a machine, and wondered if in these conditions the interrogator C would pass a wrong judgement with the same frequency as the classic imitation game. C is wrong when he guesses that the human being is A and not B: in that case the machine wins and is pronounced able to think.

Now we shall submit the imitation game and the variant proposed by Turing to a close examination. First of all we should answer the following question: what does the imitation game prove when it ceases to be a pastime to become a Gedankenexperiment? (see Box 1).

The underlying idea is that the answers of a man might be distinguishable from those of a woman, implying that the male way of thinking differs appreciably from the female one. But, although the female intelligence might be different from the male intelligence, the imitation game is based on a second assumption, i.e. that a man could nevertheless simulate the female intelligence.

However, one should not forget that the game involves three players: the identification of A and B is assigned to a judge, the interrogator, and if the judge is smart enough he can make the correct decision, with B’s help, in spite of A’s efforts to lead him astray.

This raises a problem: is it a matter of deciding if A is smarter than B or of deciding if C is smart enough? Perhaps the game is a test for C: if the interrogator is changed, but A and B are the same, the outcome of the game could change. One can also imagine to repeat the game with the same judge, playing several hands and producing a statistical evaluation of the results. Alternatively, one can imagine that there are several interrogators, each of them making a decision on the basis of the answers given by A and B to the same series of questions. Their decisions can be different, and again a statistics can be set up.

In the original version of the game the judge tries to discriminate between a man and a woman. This is a classic dichotomy, but one could imagine other dichotomic pairs of participants: a white and a black, an Italian and a Chinese, a poet and a mathematician (the sex of all of these is arbitrary)...

And the contest can be made more complicated, contrasting e.g. a Chinese mathematician and an Italian poet... From these examples it should be evident that C, who knows to what classes or categories A and B belong, does certainly decide on the basis of the answers he receives, but is also influenced by his beliefs, prejudices, experience of the world and so on.

In other words, even when the messages are exchanged by teleprinter, the answers contain elements that are not purely syntactic, i.e. internal to the language, but also, and markedly, semantic, in the sense that they are linked to the overall nature of A and B, not only to their cognitive properties. Of course the presence and the perceptible amount of such semantic elements depend on the answers, hence on the questions of the interrogator and on his expectations about the answers.

Were it possible to ask purely syntactic questions (e.g. of logical or mathematical nature) calling for purely syntactic answers, perhaps it would be impossible to distinguish a man from a woman, although recent neurophysiological discoveries seem to point to remarkable brain differences between sexes, hence to possible differences in thinking. Of course we are referring here to a man and a woman chosen at random and unknown to the interrogator.

It would ensure that the distinction between sexes could be drawn only on the basis of facts not reducible to sequences of symbols, or better on the basis of extralinguistic facts that can nevertheless leave an imprint on the sequences of symbols.

Stated differently, it exists the possibility that the differences (of experience, of social role, of biological characteristics and so on) between man and woman entail detectable differences in the answers that A and B give to certain questions, differences that cannot be made up for by imitation. That would indicate that thought cannot be isolated or abstracted completely from the rest of the human being, e.g. from the body and its needs, hence from the world. In short, if the differences between the answers given by A and B proceeded from the differences between A and B, a sufficiently smart interrogator would always make the correct decision, as B could never imitate A: to imitate A, B should be A.

3 Body and Mind

© CEPIS
Let A, B and C (the interrogator) be the participants in the imitation game. Here are some of the possible variants:

1. A man, B woman, C human (original game).
2. A machine, B human, C human (Turing’s variant).
3. A machine, C human (first variant with two players).
4. A human, C human (second variant with two players).
5. A man, B woman, C machine.
6. A human, C machine (third variant with two players).
7. A machine, C machine (fourth variant with two players).

In all the variants the interrogator C should establish A’s nature. In the first variant the interrogator knows that the players are a man and a woman, but he does not see them. In all other variants the interrogator may or may not be aware of the possible identities of A (and B, if present). Turing himself does not make clear whether in variant 2 the interrogator knows that one of the two contestants is a machine, although it is rather obvious that he should know. Consider variant 3: if the interrogator does not know that A can be a machine, he might always conclude that A is human, possibly stupid, insane or facetious. This happens all the more so in variant 4, where the mistakes made by human A to baffle the interrogator would possess the human quality of pertinence (see Box 2). Thus to make the game more interesting the interrogator should be aware of the possible nature of the competitor(s). In cases 5, 6, and 7 providing the interrogator with information concerning the possible nature of the contenders is problematic, as it is hard to envisage the questions that C might pose to conduct the examination (furthermore one could wonder whether C himself has passed the Turing test...). To sum up, these three variants are not worthy of great attention, but perhaps they confirm that the test has to do with the interrogator as much as with A and B, and even more: at least from our viewpoint only a human intelligence can judge a human (or exotic) intelligence. This seems to confirm that we consider human intelligence higher than machine’s, as much as in school examinations the intelligence (or competence) of the examiner exceeds (or should exceed) the pupil’s. I do not know if this anthropocentrism is unavoidable as it seems to me right now. In variant 7, however, we catch sight of a new perspective: humans will play the game with humans and with machines until machines are able to play like humans or even better. At that point it is plausible that humans play only with humans and machines only with machines (this happens already with chess programs and has been happening for a long time in races: horses contend with horses, cars with cars, and – let me add – men with men, women with women!). But when machines will have exceeded humans, perhaps the most qualified interrogators for all variants will be machines, and we shall have to adopt a machine-centric viewpoint...

The underlying idea is that, at least until now, it is not possible to formalize the general concept of intelligence. We might agree to assign well-defined limits to the contest between man and machine, setting up a sort of match, as it were, ruled by quantifiable parameters, e. g. – in the case of conversation – richness of vocabulary, diversity in the construction of sentences, logical correctness, frequency of metaphors... and of course duration of the competition. A problem arises immediately though: what should the conversation range be? Or should the conversation extend over any subject? It is easily understood that in the latter case no existing conversation program can pass the test if the match lasts longer than a very short time (see Box 5), while if we limit the conversation range drastically, success does not prove anything, since a real conversation between human beings is an interactive, constructive, autopoietic, arborescent, partly random and partly deterministic process that reflects the events of life closely, imitating their whimsicality and unpredictability, but also their complex logic open toward the world. As Licata puts it, "the process of human cognition is moored at the unpredictable complex boundary between a system and the environment that surrounds it."

Box 1: Some Possible Variants of the Imitation Game.
Turing has a different position: in his opinion, thought can be separated from its support. Actually the game has a sense only if C is isolated from A and B, otherwise he would distinguish the man from the woman or, in the variant, the man from the machine at once, just by looking at them. Turing extends this condition to the prohibition of considering bodily features or functions (beauty, dancing ability and so on), that would discriminate against the machine and that actually, for him, do not add a speck to the intelligence of the owner.

Hence Turing, and many others, nurtured the prejudice that well-constructed sequences of symbols are sufficient to express and recognize intelligence. Body has nothing to do with intelligence. Such mentalistic or informational reductionism is confirmed by the list of human activities that Turing judged suitable for machine implementation. Such activities were those that do not involve a contact with the outer world: chess, mathematics, cryptography, perhaps translation from one language to another. But all bodily factors were excluded. In a BBC broadcast he declared:

I am sure, and hope, that not many efforts will be made to construct machines possessing specifically human characteristics (except for intellectual ones), such as the form of the body. To me such efforts appear insignificant and their results would have approximately the same unpleasant quality of artificial flowers. It seems to me that constructing a thinking machine is an endeavour belonging to a different category.

However, in the 1950 paper Turing contradicts this position partially: by extending the activity of intelligent machines to conversation, in fact identifying thought with linguistic ability, perhaps inadvertently he opens a wide gap through which the machine comes into contact with the outer world. Actually the interrogator can ask arbitrary questions, in particular questions concerning the facts of the world (the world where we humans dwell). The link between language and world, that shows up when language ceases to be self-referential, i.e. a set of purely syntactic symbols forming an abstract formalism à la Hilbert, and becomes laid with semantics, i.e. refers to things outside the language, such a link, by virtue of which language creates, modifies, and organizes the world and is conditioned by it, now affects the machine as well (see Figure 2).

As we see, there is a certain confusion: on one hand Turing admits that human intelligence derives from the interaction with the world, hence from the body, which is the means through which that interaction takes place. On the other hand he is convinced that a discrete-state machine, the so-called "Turing machine" (see Box 2), although it does not possess neither sense organs nor actuators that allow it to interact with the outer world, can nevertheless use words like a human being and with such an ability as to cheat the interrogator.

The imitation game is based on linguistic interactions, i.e. on exterior marks of thought, the latter being inaccessible. In this sense it is a behavioural criterion: verbal sequences are there to exhibit their existence, whereas thought, from which they (might) emanate, has a much more doubtful existence (actually in the first half of the XXth century thought was banished from the reflections of psychologists and philosophers). So Turing accepted the behavioural position that held sway at those times, but in spite of this he did not deny the existence of thought, in fact he contributed to formulate a problem still lively in cognitive sciences, i.e. the relation between different levels of functioning and description of thought. Actually, among the objections he considered the following deserves attention:

May not machines carry out something which ought to be described as thinking but which is very different from what a man does?

In other words, even if the surface activity, i.e. the linguistic expression, is the same, the underlying mechanisms might be very different in machines and in humans. On the other

---

1 In the 1950 paper, Turing anticipates the critics’ objections listing and discussing nine contrary opinions: 1. The theological objection: thinking is a function of the immortal soul that men own and machines do not; 2. The “heads in the sand” objection: the consequences of machines thinking would be too dreadful, let us hope and believe that they cannot do so; 3. The mathematical objection: as a result of Gödel’s theorems there are questions to which machines cannot answer; 4. The argument from consciousness: see Jefferson’s statements in this article; 5. The argument from various disabilities: the listed physical inabilities would prevent the machine from thinking (see the considerations concerning strawberries and cream in this article); 6. Lady Lovelace’s objection: “The Analytical Engine has no pretensions to originate anything. It can do whatever we know how to order it to perform” (her italics); 7. The argument from continuity in the nervous system: as the nervous system is certainly not a discrete-state machine, it may be argued that one cannot expect to be able to mimic its behaviour with a discrete-state system; 8. The argument from informality of behaviour: since humans do not behave according to a set of rigid rules, but adopt a flexible behaviour, men are not machines; 9. The argument from extrasensory perception, or ESP (Turing declares that this argument is to his mind quite a strong one): since the statistical evidence in favour of ESP in man is overwhelming, man and machine would be in quite different positions as to the answers, and identification would be easier.
As I have mentioned cursorily, the "machine" that Turing had in mind is a finite state digital machine, i.e. a particular computer, or better an abstract prototype of a concrete computer, called "Turing machine" (TM). It is an ideal device that operates on a tape of unlimited length, divided into cells. The tape can move to the right or to the left one cell at the time. Thanks to a writing and reading head, a symbol from a finite alphabet can be read, written or cancelled on each cell. The TM can carry out any operation that can be described through an algorithm and in this sense is a universal machine. Turing however proved that the TM is not able to find a solution to all mathematical problems and linked that discovery to Gödel’s theorems.

According to Turing’s biographer, Andrew Hodges, the fact that the TM communicates with the world only through the exiguous channel of the program provides an indication of Turing’s ideal lifestyle: "a life where he would be let alone in a room of his own, interacting with the outer world only by means of rational reasoning." This remark confirms a possible direct influence of personal experience on the creation of scientific ideas. Against this interpretation, however, in a passage from his Ph.D. thesis at Princeton (1938) Turing mentions a very important human ability by means of which a mathematician distinguishes the topics of some interest from all others. According to Turing, the existence of such an ability contradicts the widespread opinion that the only activity of a mathematician is to determine the truth or falsity of propositions.

The ability referred to by Turing depends on the contact with reality, hence machines cannot possess it. Also the following passage from the 1950 article points to the importance that Turing ascribed to the contact with the world:

The popular view that scientists proceed inexorably from well-established fact to well-established fact, never being influenced by any improved conjecture, is quite mistaken. Provided it is made clear which are proved facts and which are conjectures, no harm can result. Conjectures are of great importance since they suggest useful lines of research. Hence it seems that Turing oscillates, without making up his mind, between an ideal and cherished self-reference of language, in particular of the logical and mathematical linguistic world, and a communicative and active openness of language toward the real world. A relevant example of self-reference in mathematics is offered by the so-called "Hilbert program". At the beginning of the XXth century, the German mathematician David Hilbert had claimed the possibility of building the whole mathematics by means of the axiomatic method, using a finite number of starting propositions (axioms) and a finite number of inference rules. The mathematical objects would then be defined in an implicit and abstract way, without any semantic reference to the outer world. In other words, Hilbert abstained deliberately from "defining" mathematical objects, that would be "named" and then connected to each other so as to be characterized by their mutual relations. Then, using the rules, the theorems would be derived from the axioms through a mechanical procedure.

But in 1931 logician Kurt Gödel proved two theorems that put very precise limits to Hilbert’s program. The first theorem asserts that a formal theory capable of expressing elementary arithmetic cannot be both consistent and complete. In particular for any consistent formal theory that proves certain fundamental arithmetic truths, there is a statement that is true but not provable in the theory. The second theorems proves that it is impossible to prove the consistency of the theory within the theory itself.

These theorems do not so much imply a defeat of mathematics or human reasoning, but rather a failure of Hilbert’s hope to isolate mathematics from any external reference: a mathematician does not restrict himself to symbol manipulation, he also considers the metamathematical context and the practical importance of problems, makes choices and detects connections between problems and seemingly far-off results.

Confronted with an undecidable statement, for instance, the mathematician resorts to semantics to extend the formal system into a larger one that in turn contains an undecidable statement, thus requiring a further extension, and so on. Only very poor formal systems, those unable to formalize arithmetic, are consistent and complete, while those just a little more powerful offer an indefinite semantic openness toward the world. This in at variance with the notion that mathematics is a finite tautology, i.e. a mechanical explicitation of true propositions (theorems) implicit in the axioms.
In 1936 Turing published On Computable Numbers, with an Application to the Entscheidungsproblem, an article that illustrated the atomic nature of computational procedures by means of the Turing machine, TM, that shows up in a proposition, the so-called Church-Turing thesis, according to which all brain processes derive from a computable substrate (this is an unduly simplified version, for a discussion and other versions see Hofstadter). As a consequence, man’s thought processes would be reproducible by means of a finite automaton like the TM. This proposition is the basis of the strong version of AI and finds its justification in the finiteness of human memory. In 1969, however, Gödel discovered a flaw in Turing’s argument, that undermined his finitistic vision of human computation. According to Gödel, the human mind is not static, but develops ceaselessly during its functioning. Although at every stage of its development the number of distinct states is finite, there is no reason why this number should not diverge in the course of that development. At the beginning the mind is based on a given finite number of axioms (that could correspond to the initial abilities specified by genetic information), and then, thanks to the perturbations generated by the interaction with the environment, i. e. thanks to the information supplied by experience, the mental system grows more and more complex and eludes our formal analysis, just as a sufficiently powerful system, as specified above, eludes every axiomatic reductionism.

What characterizes the human mind is the ability to produce novelties, to break symmetries, to learn, briefly to perform all those creative functions so well described by Gregory Bateson in his essay on the logical categories of learning and communication. The mind-environment interaction is generative and dynamic and cannot be confined in an ultimate description of mind by means of one or more TM. AI possesses closed logical structures and is successful within specialized microworlds or in tackling toy problems, whereas the mind displays an open logical structure, constantly modifies the rules of the game and adapts to the changing interactions with the (macro)world.

In fact, also classical physics, apparently possessing a closed structure expressed by the inexorable development of phenomena according to adamant eternal laws, is unable to account for novelties, information emergence, and history. Only the advent of new concepts and perspectives, such as complexity, emergency and information, has provided an evolutionary, historical and open picture of physical reality as filtered by our cognition. In all this one can recognize the interlacement between physics and mind that Gregory Bateson had described in philosophical and qualitative terms in his "ecology of mind".

One should not conclude, however, that Turing was so naive as to believe that one TM or even a family of TMs could be an adequate model of the human mind. Both in the 1936 article and in a 1948 report, Turing states that the machine organization and functioning should be liable to undergo changes. The first functional change is represented by oracles, devices that can influence the computation of a TM from the outside. There is a connection between oracles and natural intelligence: the program recorded in the genome is progressively modified by experience. Referring to the initial organization and the ability to obey programs, Turing says that discipline alone is not sufficient, intelligence demands initiative, too:

Intelligent behaviour presumably consists in a departure from the completely disciplined behaviour involved in computation, but a rather slight one, which does not give rise to random behaviour, or to pointless repetitive loops.

Hence Turing realizes that the TM model is too poor to describe cognitive processes: paradoxically, the machine should be able to make mistakes: if it is infallible it cannot be intelligent. Actually, a mathematician undergoes a long training, during which in general he makes many mistakes. Why things should be different in the case of a machine?

Such proneness to errors allows to avoid what is repetitive and mechanical in certain programs that aim at imitating the conversational skills of humans. While the mistakes made by machines in answering the questions of the Turing test can be arbitrarily gross and blatant, in the errors made by humans it is always to be found a certain amount of relevance, a wider and fuzzier notion than correctness. Thus relevance pertains also to some types of error: it is only important that the wrong answers do not give the impression of arbitrariness or insanity. The notion of relevance can help us grasp the (vague) idea of understanding better that the (absolute) notion of exactness (see Box 5).

Box 2: The Machine, Hilbert and Gödel.

4 Turing’s Variant
Consider now the variant proposed by Turing, in which A is a machine and B a human being. In this case too it is
not very clear what the meaning of the game might be. One can imagine that repeating the test several times, i.e., playing several hands, sometimes C is mistaken and sometimes he is right in his judgement, in other words sometimes the machine passes and sometimes it fails the test. It is easily understood that, if the questions can be arbitrary, sooner or later the machine will make revealing mistakes: it is enough to ask questions about the world at large, exceeding the machine’s competence area. But this holds true also for humans. Perhaps, as in all games, limitations and conditions should be introduced in the first place. Assuming now that we have settled duration, constraints etc. of the test, we can wonder whether passing the test is a sufficient condition for possessing intelligence. Or is it a necessary condition? Or is it a simple display of ability, a pastime, at most a subject for meditation? To get some cues in this respect, let us ask similar questions about human beings, to whom in general we do not hesitate to grant intelligence.

**5 Human Intelligence**

Assume that a human is subjected to a particular test, that consists in playing several chess games. Can we say that if he wins more than fifty percent of the games he is intelligent? And that if he wins less than fifty percent he is not intelligent or is less intelligent? Few people, I believe, would subscribe to such radical conclusions, most would simply agree that the subject is good or bad at chess relative to his opponent. Now assume that the subject submits to a more elusive test, consisting in writing a story. Also in this case, as in the previous one, success is not a sufficient condition to proclaim the subject as intelligent, neither is it a necessary condition. In this case, however, to pass judgment about success would be much more difficult...

Many people are considered intelligent (on the basis of circumstantial evidences, such as success in professional career or ability in establishing satisfactory personal relations or skill in scientific research...), and yet they cannot play chess or write stories. Perhaps they could engage in sketching a story, but success (or failure) would not add (or subtract) a speck to their intelligence, it would only be another circumstantial evidence, valid for those who think that stories are important: once more the point of view of the judge is stressed. Conversely, many people who play good chess or write excellent stories often prove rather stupid in many practical situations.

The point is that human intelligence is broad spectrum, so to speak, manifold, elusive and diversified (see quotation from Descartes in Box 3), it originates, grows and modifies through communication and more generally through that equally manifold and elusive phenomenon that is life. It follows that human intelligence is strictly linked with the body and with its immersion in the environment, moreover, it is a product of evolution, biological first and then cultural. In addition, we observe that the enormous number of human beings that live, act and communicate in each instant of time makes it something of a problem the use of generic terms like "man", "woman" or of fuzzy properties like "human intelligence".

Machine intelligence, whatever its meaning and significance, at least for the moment is very narrow spectrum, is aimed at particular tasks and in general communicates with the environment only through the thin cord of the program. It does not prolong into the outer world and does not possess the diachronic dimension of evolution.

The reader has the perplexing impression that in his article Turing considers his proposal a game (actually a game imitating the imitation game) rather than an indisputable intelligence evaluation: only later, after Turing’s death, people began to speak diffusely of "Turing test". Also, in spite of the importance of the subject and the in-depth investigation of it, the reader cannot avoid the impression that Turing’s attitude is somewhat playful and that now and then he indulges in some curious digression (e.g., see the answer given to the objection from the extrasensory perception in note 1 above).

Curiously enough, there is also a calculation error: when the interrogator asks the (human or artificial?) subject to sum 34957 and 70764 the answer is 105621 instead of correct 105721. What is the meaning of this miscalculation? Or is it an oversight?

**6 Cognitive Projection and Consciousness**

Let me consider again the interrogator’s position. If the test concerns C, as I have hinted, then the experiment is not as much about machine intelligence as about our attitude regarding the (linguistic, in this case) manifestations of intelligence, be it simulated or "natural".

Such a change in perspective is supported by the events concerning some programs produced later, e.g. Eliza by Joseph Weizenbaum (see Box 4). In any case one should keep in mind that the machine does not simulate man’s thought, rather it simulates a
I owe to Bianchini two interesting quotations from Descartes and Leibniz that anticipate very aptly some of the deepest topics of today's AI. Descartes' passage describes some criteria to differentiate human intelligence from AI, in particular draws attention to linguistic behaviour. One cannot but notice the kinship with Turing's arguments. Leibniz seems to hint at the possibility that there exist two types of explanations, one for mechanical phenomena and one for phenomena relative to intelligence (or mind or feeling). Such a double explanation does not seem to have an ontological nature, but only an epistemological one, and is reminiscent of Bateson's double explanation of physical and mental phenomena, respectively. Taken together, the two passages provide a concise formulation of a central problem in AI, i.e. the evaluation of explanation models for intelligent action.

"... but if there were machines bearing the image of our bodies, and capable of imitating our actions as far as it is morally possible, there would still remain two most certain tests whereby to know that they were not therefore really men. Of these the first is that they could never use words or other signs arranged in such a manner as is competent to us in order to declare our thoughts to others: for we may easily conceive a machine to be so constructed that it emits vocables, and even that it emits some correspondent to the action upon it of external objects which cause a change in its organs; for example, if touched in a particular place it may demand what we wish to say to it; if in another it may cry out that it is hurt, and such like; but not that it should arrange them variously so as appositely to reply to what is said in its presence, as men of the lowest grade of intellect can do. The second test is, that although such machines might execute many things with equal or perhaps greater perfection than any of us, they would, without doubt, fail in certain others from which it could be discovered that they did not act from knowledge, but solely from the disposition of their organs: for while reason is an universal instrument that is alike available on every occasion, these organs, on the contrary, need a particular arrangement for each particular action; whence it must be morally impossible that there should exist in any machine a diversity of organs sufficient to enable it to act in all the occurrences of life, in the way in which our reason enables us to act."

Descartes, Discourse on the Method of Rightly Conducting the Reason, and Seeking Truth in the Science, Chapter 5.

"Moreover, it must be confessed that perception and that which depends upon it are inexplicable on mechanical grounds, that is to say, by means of figures and motions. And supposing there were a machine, so constructed as to think, feel, and have perception, it might be conceived as increased in size, while keeping the same proportions, so that one might go into it as into a mill. That being so, we should, on examining its interior, find only parts which work one upon another, and never anything by which to explain a perception. Thus it is in a simple substance, and not in a compound or in a machine, that perception must be sought for. Further, nothing but this (namely, perceptions and their changes) can be found in a simple substance. It is also in this alone that all the internal activities of simple substances can consist."

Leibniz, Monadology, 17.

Box 3: Two Eminent Predecessors: Descartes and Leibniz.

strongly simplified model of man's thought (see Figure 3).

With respect to programs like Eliza, human beings operate a sort of "cognitive projection" similar to the psychological phenomenon of affective projection: if the external features of a given entity E and the external features of a human being U are similar, we feel prompted to operate a "completion", i.e. to ascribe to E all the properties of U, in particular the psychological, affective and cognitive ones. This happens all the more so when E is a human being, too, i.e. when it owns all the external features of mankind: the projection that we carry out in this case is so frequent as to go unnoticed, and consists in ascribing to the other the same feelings, logical and argumentative abilities and so on, that we have. It is only by virtue of this sort of identification that we overcome the so-called problem of other minds and persist in our tenacious endeavour to communicate without really knowing what happens to our messages once they leave us to enter the interlocutor.

The problem of other minds is similar to the problem of consciousness: since I cannot penetrate into the other person, since I have no means to know if he or she owns consciousness and other characteristics, I generously and trustingly credit him or her with the same faculties that I own. Turing concludes by stating that there is no reason why a computer should be treated in a different way.
The computer program *Eliza*, written in 1966 by Joseph Weizenbaum (at that time researcher at MIT) derives its name from the female protagonist of *Pygmalion*, a play by George Bernard Shaw. In the play professor Higgins bets that, acting on her phonetics and language, he can transform the shabby girl of the people into a sophisticated lady speaking with an upper-class accent. Actually *Eliza*, that can hold a conversation in natural language with a human user, can be improved gradually by a "teacher" that plays the role of Higgins. The dialogue between computer and human takes place by teleprinter: the user types a sentence and *Eliza*, in turn, analyses it and replies typing a sentence. Here is a fragment of a dialogue between *Eliza* and a young patient (patient's sentences are in italics):

*Men are all alike.*  
In what way?  
They're always bugging us about something specific or other.  
Can you think of a specific example?  
*Well, my boyfriend made me come here.*  
Your boyfriend made you come here?  
*He says I'm depressed much of the time.*  
I am sorry to hear you are depressed.  
*It's true. I'm unhappy.*  
What would it mean to you if you got some help?  
*Perhaps I could learn to get along with my mother.*  
Tell me more about your family.  
*My mother takes care of me.*  
Who else in your family takes care of you?  
*My father.*  
Your father.  
You are like my father in some ways.  
What resemblance do you see?  
*You are not very aggressive but I think you don't want me to notice that.*  
What makes you think I am not very aggressive?  
*You don't argue with me.*  
Why do you think I don't argue with you?  
*You are afraid of me.*  
Does it please you to believe I am afraid of you?  
*My father is afraid of everybody.*  
What else comes to mind when you think of your father?  
*Bullies.*  
Does that have anything to do with the fact that your boyfriend made you come here?

Weizenbaum said that *Eliza* provided a "parody" of the responses of a nondirectional (Rogerian) therapist in an initial psychiatric interview. *Eliza* functioning was based on simple parsing and substitution of key words into standard phrases. Depending on the initial sentences typed by the user, the illusion of a human interlocutor could be instantly dispelled or could continue through many interchanges (as in the example above). *Eliza* was sometimes so convincing as to catch up people emotionally for several minutes before the machine's lack of genuine understanding became obvious.

Briefly, the program conforms to the following procedure: *Eliza* looks for some key word in the patient's sentence; if it finds one, it transforms the sentence according to a rule associated to that key word, otherwise it makes a generic remark or, under certain conditions, repeats a previous transformation (actually the program is much more complex, but it is out of place to go into detail here). *Eliza* may give the interlocutor the impression of being an understanding but rather sluggish therapist: he puts vague questions and talks very little, as is seen from the dialogue above. *Eliza* does not possess even a tiny spark of intelligence: the program simply looks for terms like "mother" or "depressed" and then digs up an appropriate question from an archive. If this strategy does not work, *Eliza* produces a generic sentence trying to start the conversation anew. Almost all conversational programs are based on similar principles (see Box 5). Actually what seems a dialogue is a monologue by the human interlocutor, who features both questions and (indirectly) answers, projects a meaning on the answers and in so doing behaves as an "animist". *Eliza* had an enormous success, and those who conversed with "her" often felt relief after a session. Kenneth Colby, a psychiatrist, declared that in a few years similar programs would be written to be utilized by therapists in their practice, and himself wrote a program, *Parry*, that simulated the linguistic behaviour of a paranoic and was able to deceive several psychiatrists about its identity. It should be noticed that simulating a psychotic is much easier than simulating a sane person. Occasionally dialogues between *Eliza* and *Parry* were organized! Weizenbaum, however, was much troubled about the success of his program, and *Eliza* was one of the main reasons why he took a stand against artificial intelligence.

**Box 4: Eliza the "Psychotherapist"**.
A contrary opinion was expressed by Michael Polanyi (1891-1976), a chemist, philosopher and economist of Hungarian origin, who declared that a machine is a machine, while a human mind is a human mind and there is no fact from experience that can change this a priori. Polanyi, however, did not elucidate the difference between machine and human mind, which is the core of the whole matter: if we remain at the surface, everything seems clear, but if we go in depth, then the difficulties begin.

This is reminiscent of what St. Augustine said about time: "If nobody asks me, I know what it is, if I have to explain, I don't know any more". This sort of cognitive presbyopia concerns language as well: if we are satisfied with a certain vagueness associated with common sense and everyday practice, everything is fine, but if we insist on taking all linguistic meanings and exceptions into account and on giving extremely accurate and all-encompassing definitions, we run into troubles. Perhaps one should be satisfied with a certain approximation, without seeking a risky precision. Enough is as good as a feast. Grasp all, lose all.

Also the great British neurophysiologist and neurosurgeon sir Geoffrey Jefferson (1886-1961) had a skeptical position, as expressed by the following objection, quoted by Turing:

"Not until a machine can write a sonnet or compose a concerto because of thoughts and emotions felt, and not by the chance fall of symbols, could we agree that machine equals brain – that it is, not only write it but know that it had written it. No mechanism could feel (and not merely artificially signal, an easy contrivance) pleasure at its successes, grief when its valves fuse, be warmed by flattery, be charmed by sex, be angry or depressed when it cannot get what it wants."

To this Turing objects:

"This argument appears to be a denial of the validity of our test. According to the most extreme form of this view the only way by which one could be sure that machine thinks is to be the machine and to feel oneself thinking. One could then describe these feelings to the world, but of course no one would be justified in taking any notice. Likewise according to this view the only way to know that a man thinks is to be that particular man. It is in fact the solipsist point of view. It may be the most logical view to hold but it makes communication of ideas difficult. A is liable to believe "A thinks but B does not" whilst B believes "B thinks but A does not." Instead of arguing continually over this point it is usual to have the polite convention that everyone thinks."

And he remarks that the imitation game with only one candidate is often practiced in oral examinations to ascertain whether the pupil has learned something by rote or has really understood it. If a sonnet-writing machine were able to pass an oral examination about poetic composition with the same self-assurance and lucid ability to justify its answers as shown by a student, it would be difficult to regard the machine as merely artificially signalling these answers or to consider it as an easy contrivance. In short, a verbal performance that could not be distinguished from that of a human being would urge us to accept the test validity. As to consciousness, however, Turing adds:

"I do not wish to give the impression that I think there is no mystery about consciousness. There is, for instance, something of a paradox connected with any attempt to localise it. But I do not think these mysteries necessarily need to be solved before we can answer the question with which we are concerned in this paper."

7 The Role of Verbal Language

"And actually all the abilities that a shrewd human storyteller, no matter how expert in characters, can detect in an animal or ascribe to it, are after all nothing but mere suppositions to which only our immediate anthropomorphism lends some plausibility. Between you and me: how can we comprehend the thoughts of a brute, the real sense of its acts, even if we adopt the human meanings of those terms? A man in front of another man has at least a convention, at all events linguistic, on the basis of which he evaluates the other's attributes; but to extend that convention to the animals would be no less than arbitrary."

Tommaso Landolfi, The two old maids

According to some, Turing's verbal criterion is not sufficient for attributing intelligence and, for that matter, is not even necessary: an entity can be intelligent without being able to speak, and actually this is the case for higher animals. Also in human beings communication is by no means only linguistic, although in the West we have privileged verbal communication and consider the other communication forms vague and ambiguous, but this could be a consequence of our cultural tradition. What matters is to be able to communicate in some way or the other: an intelligent entity that were not able to communicate at all would be reminiscent of Berkeley's tree that falls in the forest without anybody around to hear its crash... The verbal criterion is anthropocentric to the utmost, but it is rather evident that certain humans would not pass the test, while some non-intelligent machines could pass it (inducing a judge into error for at least five minutes, as in the case of Eliza). This seems to weaken the significance of the criterion greatly. Hence any attempt at separating mind from body, attributing intelligence only to the former, raises a certain number of problems that are typical of the functionalist approach to artificial intelligence and have caused the failure of its strong version.

Such problems can be summarized by means of a few crucial questions:

Can it exist an intelligence without life and without other attributes of life, e.g. emotions and consciousness? Can it exist a mind without communicative interaction with the others? Can it exist a self-referential language, without any form of material support or existence, therefore without any interaction with the world? And, perhaps the most crucial question: can thought exist without experience (experience being rooted in body)?

These questions exhibit a clear anthropocentric nature: we choose our cognitive activity as a reference model,
At least at first sight the Turing test is extraordinarily simple, but for some decades nobody took into consideration the possibility of implementing it: everybody, including Turing himself, considered it only a thought experiment. Sure, in his 1950 article Turing had predicted a bright future for machines, albeit in vague terms:

_I believe that in about fifty years’ time it will be possible, to programme computers, with a storage capacity of about 10^9, to make them play the imitation game so well that an average interrogator will not have more than 70 per cent chance of making the right identification after five minutes of questioning. The original question, “Can machines think?” I believe to be too meaningless to deserve discussion. Nevertheless I believe that at the end of the century the use of words and general educated opinion will have altered so much that one will be able to speak of machines thinking without expecting to be contradicted._

Apparently, however, the researchers that followed were less optimistic and did not venture into performing the test. In 1991, however, programmer and businessman Hugh Loebner, from New Jersey, set up a US$ 100.000 prize to be awarded to the first program that judges cannot distinguish from a real human. Once this is achieved, the annual competition will end. Feeling that this goal is pretty remote, Loebner decided to award a $ 2000 prize to the best competitor every year. The contest has been followed with great interest since, although no program ever came close to the goal, so that the competition more or less turned into a publicity event. In 1966 the match was sort of a surprise: among all thinking, or allegedly thinking, machines an impostor had crept, program HeX, written by Jason Hutchens, a doctoral student at the University of Western Australia. And, to everybody’s dismay and consternation, HeX won the $ 2000 yearly prize.

HeX, as Hutchens later stated, had nothing to do with AI, in fact was about as smart as a Mr. Coffee. All Hutchens had done was to write a set of questions, answers and glib statements that the computer would emit on cue. He had done no attempt to write a program using AI techniques, in fact had avoided them on purpose to demonstrate that the Loebner competition was a waste of time for AI. Although he declared that his only reason for entering that “annual demonstration for the media” was for “fame and fortune”, it cannot be maintained that Hutchens had violated any rule in the competition, mainly because the Loebner prize has no rule on the content or intent of a competitor (this vagueness, in a way, reflects the vagueness of the Turing test). On the other hand, Hutchens’ competitors, who employed a vast assortment of AI techniques, wrote programs that could hardly be considered intelligent. Here is a short sample of a conversation between an interrogator and a losing participant named Fred revision 16.43:

Fred: Welcome judge 4.
Interrog.: Hello. What’s the weather outside?
Fred: I haven’t been outside much today. I am fond of asparagus myself.
Interrog.: How old are you?
Fred: I’m about 30.
Interrog.: About?
Fred: I can talk about Star Trek.
Interrog.: No let’s talk about you. Exactly how old are you?
Fred: Right you are batman.

HeX was not much better, as shown by the following sample:

HeX: I go to the University of Western Australia, which is one of the nicest universities in Australia. It is situated on the Swan River in Perth.
Interrog.: What does "situated" mean?
HeX: I have absolutely no idea what "situated" mean does.
Interrog.: Um. Are you tired?
HeX: Everyone knows that I am tired. And I am seldom boastful. What do you think of this Loebner contest thing?
Interrog.: It’s off to a rip-roaring start, huh?
HeX: Hehehe - that's supposed to be a question??? Have you seen any good films lately?

Consider now the following conversation: is the contestant a computer or a human being (Turing test for the reader...)?
Given the level of competition between computers, who cares if Hutchens cheated? As the transcripts show, the computers are far off the mark from human intelligence, and one gets the impression that all conversations are disconnected, absurd, not relevant, i.e. artificial. Sure, programs will improve their performance eventually, but as long as they stay secluded in their virtual world they won't be able to answer questions on the Swan River or on weather: what does the Swan River mean to a creature that has never touched water or felt thirsty? Our human thoughts are a delicate tangle, an elusive dance between mind and body, whereas the Turing test relies on language alone and risks to isolate thought in a purely mental, abstract space. With enough questioning any intelligence not having experience of the world can be unmasked, revealing the qualitative difference between man and computer. Such difference stems essentially from man's evolution and immersion in the environment, briefly from the presence of the body. The body confers a sense to what we do, hope, expect or fear: we are semantic entities. In the light of this, one becomes aware of how inadequate the expression "artificial intelligence" is. One should stress the adjective "artificial" much more than the noun "intelligence".

with the preconception that any proposed alternative is worse or flawed. At the same time, the questions contain vague, indefinite, inaccurate terms, like life, consciousness, experience. To these terms, however, we, as human beings, are able to give an operational meaning, and in front of them we adopt a cautious attitude, consisting in postponing precise definitions and using intuition. Transferring such concepts to machines, that cannot benefit from the operational approach or from intuition, would require unambiguous and flawless definitions. It is obvious that under this respect machines are in an unfavourable condition.

Ludwig Wittgenstein raised similar questions, too: can we consider language as a mere game? Or has it a connection with real life? It is rather curious that even today some thinkers are inclined to answer the first question in the affirmative, forgetting the concrete basis of language and the inextricable tangle that links it to the world through human activities. Also the consequences of Gödel's theorems, hinting at an indefinite openness of mathematics and at its connections with the world, should make a similar openness of language and mind toward the world evident (see Box 2).

Turing oscillates between two positions: one is favourable to the dematerialization of intelligence, which is so well expressed by the leading character in the book The Small Back Room by Nigel Balchin: "It's a great pity when you come to think of it that we can't abolish the Navy, the Army and the Air Force and just get on with winning the war without them." The other position, illustrated by Turing himself in the article, acknowledges the importance of the link with reality for the machine to be complete. After listing various properties that, according to some, if not possessed by a machine would make it unable to think (such as: be kind, resourceful, beautiful, friendly, have initiative, have a sense of humour, tell right from wrong, make mistakes, fall in love, enjoy strawberries and cream, make some one fall in love with it, learn from experience, use words properly, be the subject of its own thought, have as much diversity of behaviour as a man, do something really new), Turing writes:

"The inability to enjoy strawberries and cream may have struck the reader as frivolous. Possibly a machine might be made to enjoy this delicious dish, but any attempt to make one do so would be idiotic. What is important about this disability is that it contributes to some of the other disabilities, e.g., to the difficulty of the same kind of friendliness occurring between man and machine as between white man and white man, or between black man and black man."

In conclusion, one could say that the so-called Turing test is a thought experiment whose practical significance, as far as machine intelligence is concerned, is almost nil. It is not a necessary condition nor a sufficient one. Its importance is rather to be found in the great quantity of reflections it has elicited on the concept of intelligence, and also of consciousness, computation etc. The very hesitations shown by Turing between a purely syntactic view of intelligence and the idea...
that intelligence is rooted in the body and in the world are useful for appreciating the limits of the strong version of AI. Turing’s optimism about the future of AI has proven to be unjustified or better ill-directed: today, in 2009, machines do a lot of extraordinary things, but do not do what Turing had imagined (see Box 5). Increasingly, machines interpret the role of essential components of the cognitive (as well as active) symbiosis with humans, but they have not reached a degree of autonomy and competence about the world sufficient to enter a plain dialogue with a member of our species.

Translation by the author

References
Selected CEPIS News

Fiona Fanning

43rd CEPIS Council and President’s Strategy Event

44 delegates from 27 countries gathered in Brussels recently for the 43rd CEPIS Council and the President’s Strategy Meeting. Dr. Vasile Baltac of ATIC (Asociatia Pentru Tehnologia Informatiei si Comunicatiilor) Romania, was inaugurated as the new CEPIS President during this meeting; we welcome Dr. Baltac to his new role. The President’s strategy meeting followed Council; this highlighted the achievements of CEPIS on its 20th Anniversary and discussed the future direction of CEPIS within Europe as outlined in the CEPIS Strategy paper 2010-2013.

A number of new focus areas emerged from this including Green ICT, Women in ICT and Software Quality. If you are interested in these topics and would like to be involved in European-level actions, please contact your national computer society to express your interest in working with CEPIS.

Dr. Baltac looks forward to working with CEPIS Members to further develop the new strategy over the course of his term.

European e-Skills 2009 Conference: Fostering ICT Professionalism

The European Commission and the European Economic and Social Committee with the support of CEPIS organised the European e-Skills 2009 conference: Fostering ICT Professionalism on November 20th. The conference attended by more than 130 people across greater Europe, was an opportunity for CEPIS to communicate and highlight the importance of ICT Professionalism. Key decision makers in business, academia, professional bodies and government recognised the role Professionalism can play to ensure Europe is innovative and competitive.

Vasile Baltac, the new CEPIS President was a keynote speaker at the event and Declan Brady, CEPIS Vice President presented the CEPIS Taskforce Professionalism report at the conference. This was received with great interest by panellists and the audience. We take this opportunity to congratulate the CEPIS Taskforce on Professionalism for their work over the last two years and Jorg Ruegg who kindly accepted to facilitate the discussions and validate the conclusions of the group.


EQANIE – Auditors needed!

EQANIE, the European Quality Assurance Network for Informatics Education (EQANIE), follow-up to EURO-In project, is seeking auditors to carry out EQANIE accreditation visits. Interested parties are invited to fill out a CV form and email them to <info@cepis.org>.

Candidates should hold a leading position in academia or in industry/professional practice in one of the disciplines of informatics. Proficiency in the English language is also required, in addition to other languages. Experience in accreditation of informatics degree programmes will be considered an advantage.

Please email <info@cepis.org> for more information.