Winter 2009

Case Study: Implementing Tools for Software Quality Assurance

James Brennan
Regis University

Regis University
College for Professional Studies Graduate Programs
Final Project/Thesis

Disclaimer

CASE STUDY: IMPLEMENTING TOOLS FOR SOFTWARE QUALITY ASSURANCE

A PROJECT

SUBMITTED ON 1ST OF DECEMBER 2009

TO THE DEPARTMENT OF INFORMATION TECHNOLOGY

OF THE SCHOOL OF COMPUTER & INFORMATION SCIENCES

OF REGIS UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS OF MASTER OF SCIENCE IN

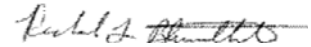
SOFTWARE DEVELOPMENT

BY

APPROVALS

Dan Likarish, Project Advisor

Shari Plantz- Masters



Richard L. Blumenthal

Date of Submission: 12/11/2009

Acknowledgements

This thesis is dedicated to my family who supported and encouraged me. I also thank my advisor Dan Likarish for his editorial advice and patience.

Abstract

Test tools have become ubiquitous in the practice of Software Quality Assurance. Every year, many millions of dollars are spent on specialized software to help manage requirements, test cases, or defects—or to automate test execution. But, after purchasing the software, many companies fail to successfully implement the tools. Given the cost of most test software, it's not surprising that companies often make a second attempt at using the tools.

This thesis describes a second-chance effort to use a suite of tools to manage various aspects of the test process. It examines some of the goals driving the adoption of test tools, as well as some of the challenges impeding it. It outlines a strategy, follows its execution, and describes the results. It looks at how tools are configured to align with the test process, and how the test process adapts to accommodate the tools. Finally, this study looks at how tool adoption is affected the opinions and attitudes of the test team, developers and management.

CASE STUDY: IMPLEMENTING TOOLS FOR SOFTWARE QUALITY ASSURANCE

A PROJECT

SUBMITTED ON 1ST OF DECEMBER 2009

TO THE DEPARTMENT OF INFORMATION TECHNOLOGY

OF THE SCHOOL OF COMPUTER & INFORMATION SCIENCES

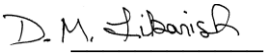
OF REGIS UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS OF MASTER OF SCIENCE IN

SOFTWARE DEVELOPMENT

BY

APPROVALS



Dan Likarish, Project Advisor

Shari Plantz- Masters

Richard L. Blumenthal

Date of Submission: 12/11/2009

Table of Contents

Case Study: Implementing Tools for Software Quality Assurance

Chapter 1: Project Definition

Software development is a competitive industry in which Software Quality Assurance (SQA) plays a critical role. "Typically, more than 50% percent of the development time is spent in testing" (Pan, 1999). Therefore, improving an organization's SQA is rightly seen as an opportunity to make the company more competitive by shortening release cycles, enabling greater functionality or improving quality.

One way to improve SQA is to use software tools designed to optimize, automate and measure the test process. Teams can develop their own tools, use open-source software, or purchase tools from a commercial vendor. But, many organizations have found that after acquiring a tool they are unable to use it effectively and the tool is eventually abandoned.

The organization which is the subject of this study had been trying to use a suite of commercial test tools. But the tools had not gained much acceptance or sustained use. This case represents an effort to restart the effort to configure the tools, align the test process, and use the new *framework*[1] successfully. By studying this case, I examine some of the issues that contribute to the success or failure of adopting and using new tools for SQA.

[1] While a *framework* often refers to a specific technical implementation, here I use it to describe a collection of tools and concepts used to facilitate software testing.

Case

Product

TrackWear[2] provides hardware, software, and services for monitoring a child's location using Global Positioning Satellite (GPS). Customers can monitor one or more children with a private system using TrackWear's software, or they can pay TrackWear for monitoring services. The child wears or carries a small, tamper-evident device that sends telemetry to a receiving station using a cellular connection. The telemetry data includes the child's precise coordinates and time of day.

The software allows caregivers to create schedules and designate "red zones" and "green zones" that apply to a child or group of children. Green zones represent areas or specific locations that the child is allowed to enter, or must enter; red zones are areas that the child must not enter. Scheduling let's users apply different rules for different times of day. In addition, the system can be configured to alert the caregiver if the child doesn't comply with the rules.

Life Cycle

TrackWear's software development team was following a lifecycle they characterized as a "Waterfall/Agile hybrid". The test team had decided on an informal test process as the most efficient way to satisfy stakeholders and assure quality. While this process was satisfactory in early development, it could not address the growing size and complexity of the software. Regression testing began to take longer and longer, with no visible increase in quality.

---

[2] Details including the name and precise business model of the organization have been obfuscated to respect the legal and proprietary rights of the subject.

Furthermore, since the test process did not produce formal metrics, it was difficult for management or the testers themselves to identify the causes of the problem or propose a solution.

Tools

The company had purchased Compuware's QACenter, a suite of tools designed to enable test automation, optimization of manual testing, requirements traceability and a host of other processes and activities. However, the tools had not been used in some time. Instead, manual tests were kept in flat files like spreadsheets or text documents. Test execution was left largely to the individual tester. And, while TrackWear had purchased the tools in part to enable automated testing, testers complained that they had received no training, and in any case had little time to learn a new tool.

In short, none of the purchased tools was currently in use although the organization continued to pay the annual maintenance fee. Based on discussions with management, this ongoing expense was one factor driving further work on the tools.

Objective

As a consultant, I was charged with using the already purchased Compuware test tools to create an integrated solution for SQA. This would include installing and configuring the software, aligning test activities, and implementing the new framework on a pilot project. The framework would focus on three main areas of improvement: setting up the software to make it easier for SQA to provide metrics to its customers; optimizing the manual test process; and enabling automated testing. Project success did not require a wholesale migration of all activities to the new framework, only that the tools be made available and usable.

The goal of the study was to evaluate the existing process, observe the process of building the SQA framework, and analyze the results. An important part of this analysis would

be to discover any human factors that might be affecting tool adoption. In this sense, it may be possible to look at these issues as they might relate to other organizations.

Chapter 2: Research and Literature

Fewster and Graham (1999) use the term "shelfware" to describe test automation tools that are purchased then end up not being used by the organization. (p. 283) The authors (1999) describe purchasing a tool to fit the needs of the organization, outlining the "critical implementation factors in ensuring that the potential benefits from the purchased tool are actually achieved" (p. 288). These include management commitment, planning, and ongoing publicity to evangelize the change.

Patton (2006) acknowledges the problem of failed tool implementation, pointing out that while tools can make the test process more efficient, "…countless test automation efforts have been abandoned and cost their projects dearly when they went astray." (p. 250) He emphasizes the need to understand the limitations of tools vs. human testers, and warns against over-reliance on test automation to demonstrate quality. (p. 251) Hutcheson (2003) goes a step further, suggesting that instead of devoting resources to implementing a tool, in some cases it may be more effective to "…invest in learning how to use what you have better". (p. 185)

Scarpino and Kovacs (2009) describe a "second attempt" tool implementation that bears some resemblance to the case studied here. The authors describe a situation in which management had become aware that "the SQA tool was not being enabled across the organization as effectively as it could." and where "…upper management decided to employ an external consultant to evaluate the project and make recommendations." (p. 147)

In evaluating the implementation effort, Scarpino and Kovacs (2009) take a "what went wrong" approach, based on a series of interviews where the researcher noted "…how many times

each interviewee indicated an adverse response" to a particular topic. (p. 148) They also acknowledge the idea of establishing a value proposition for the tools, saying that "In order for overall acceptance to occur, the users must see some value in the change" (p. 150). Finally, the authors conclude that inadequate definition of QA processes, lack of management support, and communication failure were major constraints to establishing the value proposition for their project. (p. 161)

Cem Kaner's *Lessons Learned in Software Testing* also points out the need to establish a well defined test process before introducing tools to automate it. Says Kaner, "If your testing is confused, test tools will reinforce the confusion" (Kaner, Falk, & Nguyen, 1999, p. 98). Lewis (2005) echoes this opinion, insisting that "A prerequisite for test automation is that a sound manual test process exists" (p. 321).

Craig and Jaskiel (2002) seem to agree that teams should have an effective test strategy in place before selecting a tool to support it. However, they acknowledges that there may be an exception in the case where the team has "no processes at all in place for a certain function". In this case, the authors allow that it may be reasonable to "choose a popular tool and create your…process around that of the tool" (p. 219).

Fewster and Graham (1999) recommend implementing new tools on a "pilot project" to resolve problems and to better understand how the tool will affect the test process. They also point out that "In a small organization, there is more emphasis on the pilot and less on marketing the concept" (p. 291).

Much of the work reviewed presents a linear, top-down model of how people and institutions operate. Initiative, decisions and resource management flow from management down to teams and individuals, while activities move from inception to analysis to planning to

execution. This study leaves open the possibility of a more interactive relationship between individuals, teams and management. It also allows for the fact that the use of tools or building of frameworks is often exploratory in nature, with teams performing multiple cycles of planning, execution and review.

Chapter 3: Methodology

The study began with a detailed assessment of the SQA effort. The initial goal was to identify strengths and weaknesses, and recommend areas for improvement. These recommendations would then drive decisions about how to best implement the Compuware tools. Much of this analysis draws on the principles of context-driven software testing[1], which avoids "best practices" in favor of examining how practices work within the specific business, technological and cultural context in which they are applied. (Kaner et al., 1999, p. 261)

Based on this approach, TrackWear's SQA processes were evaluated in terms of customer service, a systematic approach to testing, and the consistency with which practices were documented and applied. Specifically, this study asked the following questions:

- What kinds of information was QA providing to developers, management, or the organization as a whole?
- How did this information affect the development and decision making processes?
- To what extent did TrackWear's approach demonstrate a scientific, measured process based on sound principles?
- How did actual practices compare to documented processes?
- How consistently were they followed?

The data used to make the assessment were gleaned from interviews, activities and artifacts related to testing.

Interviews with stakeholders

At the start of the project management, testers and developers were interviewed to understand TrackWear's *Software Development Life Cycle* (SDLC), the current state of testing, and attitudes about tools and test automation. Each subject was asked to describe the development process from requirements gathering to release and to identify areas they thought the SQA effort could be improved.

Observation of the test process

The study followed TrackWear's testers on a small project in order to document the test process as it took place on a live project. The interaction with developers and management was noted, as was the way defects were investigated and resolved, and what kinds of information the test team captured and published.

Tests

Approximately 20% of TrackWear's written manual tests were reviewed, including at least one test set written by each tester.

Source Code

The source code for a number of components in TrackWear's system was audited to evaluate code quality and testability.

Defects

A representative sample of defects from current and past projects was studied to understand what components, services or technologies presented the biggest risks. Special attention was paid to defects that escaped testing and were subsequently discovered in production. These production defects would be used to identify weaknesses or blind spots in the test process.

Tools

The study looked into the tools being used by SQA in the course of testing, how they were being used, and to what effect.

Scope

Based on the existing research, it appears that the phenomenon of tool abandonment is as common as it is expensive. Given the cost involved, it should come as no surprise that efforts to re-implement such tools are also common. This study looks at the dynamics of such a "second-chance" implementation with a view toward identifying various drivers and constraints affecting its success.

This study is not intended to be an exhaustive discussion of SQA process improvement or the use of tools in general. Furthermore, since it deals with tools that have already been purchased by the company, this work does not include an examination of the tool selection process. Finally, this study avoids restating institutional values like the need for planning, communication, and training except as they relate directly to this case.

Questions

Answers Needs of Stakeholders

How could the Compuware suite be configured and used to answer specific needs or goals of the organization? Management had purchased the tools as a comprehensive solution for test planning, test design, automation, and metrics. Once constructed, would this framework offer a convenient, accessible way to upgrade the test effort? If not, to what extent could the team make use of specific functions to improve various aspects of testing?

Tools First

Successful tool implementation was clearly not a question of process-before-tools, or vice versa. Rather, it was a matter of altering the tool *and* the process to achieve the desired result. Furthermore, since both tool and process require change to a greater or lesser extent, it seemed less important which element was driving the change.

Therefore, at least from an implementation standpoint, a "tools first" approach seems to be at least defensible. However, the question remained as to what extent this approach would lead to tool adoption. It also remained to been seen how successful the test team would be in changing their process to accommodate the tool.

Value Proposition

The process of implementing a tool once it has been purchased is often described as involving "inside sales". That is, the tool vendor sold the tool to the organization, now the organization needs to sell the tool to its members. This can be described as offering a *value proposition*. How would the pilot project help establish and socialize a value proposition regarding the Compuware tools?

There is cost associated with change. While it's relatively easy to configure a tool, changing the test process can be more challenging because it requires people to change the way they do things. At the very least, the team will probably have to learn the how to use the menus, interfaces and functions of the new software. The higher the cost of these changes is perceived relative to the benefits they promise, the less willing stakeholders will be to adopt the proposed changes. Fewster and Graham (1999) call this psychological phenomenon the *change equation*:

Change only occurs when three things are greater than the fourth.

$f(a, b, c) > z$

> …where *a* is dissatisfaction with the current state, *b* is a shared vision of the future, and *c* is a concrete knowledge about the steps to get from *a* to *b*. These three things taken together must be greater than *z*, the psychological or emotional cost to the individual of changing the way they work. (Fewster & Graham, 1999, p. 297)

This equation suggests several ways to affect the value proposition behind implementing new tools. This study is particularly interested in the extent to which management's concerns about testing could be said to represent (*a*) *dissatisfaction with the current state*. It also asks how effective the pilot project would be in developing (*b*) *a shared vision of the future* and (*c*) *knowledge about the steps to get from a to b*.

The strategy behind the pilot project was to decrease cost to the team by hiring someone to absorb a large part of the effort required to set up and learn the tools. Teams would be able to see the benefit without incurring the cost. The hope was that this would encourage testers and teams to adopt the tools.

Summary

This study examines one organization's effort to implement a technical framework for SQA. While much of the existing research assumes that there should be a *process* separate and distinct from actual behavior, this study takes a context-driven approach, exploring how specific test activities and artifacts serve SQA's customers. In this way, the study is able to address how the new tools affect testing, rather than how they fit a process. This study also recognizes that implementing a framework may be iterative in nature and require less detailed planning up-front.

Most importantly, this study explores how the Compuware tools could be used to satisfy the needs of SQA's customers. It looks at the implications of a tools-first approach to improving

the test process. And it evaluates the extent to which the pilot project and related activities could be said to affect the tools' value proposition and help drive the psychological factors affecting adoption of the tools.

Chapter 4: Test Tools and SQA

Greater demands for efficiency, flexibility, and transparency of the test process require software specifically designed for SQA—test tools.

> A testing tool is a software application that helps automate some part of the testing process that would otherwise be performed manually. In this category, we also include tools that support testing, such as some configuration management tools, project management tools, defect tracking tools, and debugging tools. (Craig & Jaskiel, 2002, p. 216)

Organizations can build their own tools, use open-source or other "free" tools, or buy a solution from a commercial vendor. They can automate one aspect of testing or create an end-to-end life cycle solution.

This chapter outlines some of the SQA tasks and activities supported by various test tools. It also discusses the difference between configuring a tool to support a process, and adapting a process to accommodate a tool, making the case that both are often necessary to take full advantage of a tool's functionality. Finally, the chapter describes several implementation strategies often used to deploy tools into the test environment.

Managing with Tools

It may be possible to test software using only spreadsheets or text documents to keep track of requirements, tests, defects and other artifacts. But, what if there is a need to link requirements to tests, or run a report showing only critical defects? What happens when there

are hundreds or even thousands of tests to be executed by a dozen testers or an offshore team? Test tools make these and other activities feasible. In addition, there are a number of activities (automated testing for example) that are simply not possible without a tool. "Test tools, especially test automation tools, almost always pay for themselves when used wisely" (Black, 2002, p. 432).

The following are some of the major activities and artifacts commonly managed using tools specifically designed for that purpose. This is by no means an exhaustive list. For the purpose of this study, tools for performance testing, security testing, risk analysis, and a number of other areas often managed with test tools have not been included. The study examines these because they are the most common and the most relevant to this case.

Requirements

In many development environments, testing is based on a formal requirements specification. "The requirements specification is a specification for a software product, program or application that performs functions in a specific environment…" (Lewis, 2005, p. 345) In a less structured environment, however, requirements may be vague, incomplete, or non-existent. "Unfortunately, many projects do not have a requirements document *per se*. In those instances, other methods or documents must be used to identify the key features and components to test" (Craig & Jaskiel, 2002, p. 107).

Requirements tools provide a structure in which requirements can be captured and managed. Using such a tool can help establish confidence that all functional requirements have been completed by development and tested by SQA.

Manual Testing

Manual testing can be an expensive and time-consuming part of the test effort. Test scripts must be created, reviewed and maintained so they are available to the team. This often means handling a large volume of tests over long periods of time.

> Even on small software projects it's possible to have many thousands of test cases. The cases may have been created by several testers over the course of several months or even years. Proper planning will organize them so that all the testers and other project team members can review and use them effectively. (Patton, 2006, p. 278)

Providing accessibility, security, and data management for this kind of test effort often requires the adoption of a specialized test management tool. "Test management tools include tools to assist in test planning, keeping track of what tests have been run, and so on. This category also includes tools to aid traceability of tests to requirements..." (Fewster & Graham, 1999, p. 8). Most test management tools let users view and edit the test inventory, manage test execution, and record results.

Test Data

Successful testing depends on test data that adequately represents real-world scenarios. "The goal of testing is to create the most realistic environment that resources allow and the risks dictate – this includes the test data" (Craig & Jaskiel, 2002, p. 208). One way to address this requirement is to generate custom data. "Generated data typically requires a tool or utility to create it…A tool, for example, may be used to create large volumes of similar data or data that varies according to an algorithm" (Craig & Jaskiel, 2002, p. 210).

Using tools to create and manipulate test data can enable testing that would be difficult or impossible using manually created data. For example, manually creating a data set that contains

a thousand unique names, addresses and phone numbers would take hours, whereas a tool can do it in seconds. "It is easy to generate large volumes of test data automatically with a restricted format and with no complicated relations between data items. Some of the earliest testing tools that were constructed were simple test data generators" (Ince, 1987, p. 67).

Test data tools can also be used to extract relational data from a production database or inject it into a test database.

Automated Testing

Test automation makes it possible to create tests that are executed by software. Once created, the tests can be run many times. This can mean faster testing, reduced tedium, or executing tests that can't be done manually. In this way, using a test tool to automate execution can "…significantly reduce the effort required for adequate testing, or significantly increase the testing which can be done in limited time" (Fewster & Graham, 1999, p. 3).

Traditionally, automated testing has been seen as a process by which existing manual tests are captured in code or scripting language so they can be run without human intervention. But, according to Kaner et al. (2002), "Automated testing is not automatic manual testing: it's nonsensical to talk about automated tests as if they were automated human testing" (p. 99). He suggests looking for opportunities to use test automation to do things tester can't do. (p. 100)

Finally, Jonathan Kohl (2007) suggests that "automated testing and manual testing need not be contradictory concepts" (Blurring Manual and Automated Testing section, para. 1). He proposes using automated and manual techniques in combination. This kind of "interactive automation", says Kohl, "bridges this gap and provides an opportunity to test applications in different ways" (Blurring Manual and Automated Testing section, para. 1).

Defects

Recording and tracking defects are tasks which are often performed by the test team. "…incident management [defect tracking] tools are used for documentation, administration, prioritization, allocation, and statistical analysis of incident reports" (Spillner, Linz, & Schaefer, 2007, p. 195). Such a tool generally includes a relational database which acts as a central repository to store defects and their attributes, allowing users to search or create reports based on various parameters. For these tasks, a specialized tool "…is practically indispensible to the test manager" (Spillner et al., 2007, p.195).

Metrics

As a business entity, SQA serves several customers: it works with developers to find and resolve defects; it provides management with information needed to monitor progress; and it acts as customer proxy in evaluating the functionality and usability of the product. Serving a diverse audience means QA must collect, manage and report a variety of metrics. "Time, cost, tests, bugs, and failures are some of the fundamental metrics specific to software testing. Derived test metrics can be made by combining these fundamental metrics" (Hutcheson, 2003, p. 125).

Many of the tools used to manage the test process also collect fundamental data relative to their function, like the number of tests executed or the number of defects discovered.

Traceability

Traceability means identifying a link between two or more artifacts that are part of the SDLC. One example is tracing requirements to tests. "Advanced test management tools support requirements-based testing. For this purpose, they allow the capture of requirements…and the linking of them with the test cases needed for validation" (Spillner et al., 2007, p. 194). In this

example, tracing tests to requirements could show the percentage of requirements for which tests have been created.

Tool Configuration

Most SQA tools allow or require a certain amount of configuration or customization to be used properly. At the very least someone will need to install the software and any required databases, web services, or other technical support. In addition, many tools require a designated administrator who can manage user accounts, upgrade the software, maintain the database, and other tasks. This can create a problem if the administrative costs have not been planned and roles assigned.

It is important to note that in general, test tools are designed to support a more or less mainstream approach to testing and the SDLC. This is especially true of the large commercial vendors who can be said to represent the "mainstream" by virtue of their market share. The extent to which an organization's SQA processes vary outside the range of the SDLC envisioned by the vendor, determines how difficult it will be to configure the tools to match the current way of doing things.

Process Adaptation

Because test tools are not process neutral, the SQA process may have to change to take advantage of the tool's functionality. For example, a tool may depend on a test artifact the team doesn't currently produce. It may define relationships between artifacts differently. Or, it may produce metrics the team doesn't use. Of course, this may mean it's simply the wrong tool. But, it could also mean that something is missing from the existing process.

This thinking goes against the grain of Scarpino & Kovacs (2009) and others who emphasize the need for a pre-defined process to determine behavior, including how tools are

used. (Fewster & Graham, 1999; Craig & Jaskiel 2002) Changing team behavior to match the tool would seem antithetical to this view. But, this process-first approach depends on a considered, correct, and systematic vision of the SQA process. Clearly, this is not always the case.

Therefore, instead of evaluating a tool based on how closely it matches the existing process, it may be more useful to look at how the tool can help facilitate changes to the process to provide better service to customers.

Implementation

While most tools are meant to be customized to meet the needs of the specific test organization, there are limits to such customization. Therefore, the process of implementing a set of test tools usually requires both customization of the tool *and* adaptation of the test process. The extent to which a team must customize or adapt depends on how closely the organization's *actual test process* is aligned with the model implied by the test software.

Here are several common approaches to implementing a tool once it has been selected.

Big Bang

In this approach, the software is installed and configured, time is taken to train the staff on the new software. Then, at a given point in time all activities are migrated to the new tool(s). This method represents a large disruption and likely delay of activities. However, it may also provide better preparation and training.

Pilot

Here, new process and tools are implemented on a small project to demonstrate functionality and prove that the tool can be used in the "live" development environment. After a

successful pilot, the organization can pursue a phased implementation or other method of bringing the tools on line.

Phased Implementation

Tools can be rolled out in phases, either by implementing tools one at a time across the organization, or by implementing a set of tools on one project or department at a time. This approach can allow more opportunity to evangelize the tools and train users.

Summary

Regardless of where tool comes from, there are associated requirements, costs and benefits that come with it. One tool may require advanced coding skills while another generates code based on user actions. Some tools require more configuration and customization that others. In addition, there is wide variation in the day-to-day administration required.

"Replacing tradition requires a culture change. And, people must change their way of working to include new tools" (Hutcheson, 2003, p. 43). But, in order to make changes to the process, teams must first understand why change is necessary. Second, members of the organization must have the ability to change. That is, they must have the conceptual knowledge and analytical skills to reframe their work. Perhaps most importantly, testers, managers and other stakeholders must be willing to put forth the effort required configure the tools, adapt the processes and implement the solution.

Chapter 5: Project History

This chapter presents a detailed view of TrackWear's test and development processes and takes a closer look at the capabilities of the Compuware tools. The goal of analyzing these tools and processes was to identify problem areas in TrackWear's SDLC that could be addressed by specific functions of the tools. This analysis was intended to lay the conceptual groundwork for implementing some or all of the Compuware software in TrackWear's development environment. The work presented here is based on the data and methodology described in the first chapter, and mirrors the assessment and recommendations presented to TrackWear's management at the conclusion of the pilot project.

History

TrackWear has a traditional management structure with executive officers at the top of the hierarchy, followed by vice presidents and directors. This project was under the supervision of the Director of Software Development. TrackWear's software development team was made up of two software architects, six developers, and four testers. Almost all of the team members had been with the company three or more years. The newest employee was a tester with one year at TrackWear.

In the two years before this study began, TrackWear had made several efforts to implement tools to help organize software development and testing. But, based on my interviews, team members were either neutral or negative in their opinions of the various tools that had been tried. In some instances, team members mentioned other tools they thought would be more useful. Others said they "would rather quit" that work with a given tool.

It's worth mentioning that the company was using *Microsoft Team Foundation Server* (TFS) for source control and defect tracking. TFS is part of a larger "life cycle" suite of tools

called *Team System* which was not in use at TrackWear, but was indicated by a few team members as the desired platform going forward. The relevant point here is that the company had invested in yet another product to integrate and manage the development life cycle, but had only implemented part of it.

Finally, there were the Compuware tools. In order to examine logs and other artifacts, it was necessary to locate the application server and databases that had hosted several of the tools. Based on this examination, testers had imported a number of manual tests from text and spreadsheet documents to the manual test tool. This had happened approximately a year before this study. There were also a small number of automated test scripts that had been created at about the same time. However, these artifacts gave no indication of a systematic or sustained effort to fully implement the tools.

TrackWear's history with tools remains somewhat mysterious. While the artifacts examined indicated activity around certain tools at certain times, interviews with staff and management did not give a clear picture of any specific problems, plans, or goals behind the activity. In addition, management made it clear that they expected the test team to quickly adopt whatever tool(s) management selected. But, team members still saw tool implementation as an open issue, one that probably would not be addressed in the near future, if it was addressed at all.

SDLC

In order to create a plan to implement the Compuware tools in TrackWear's development environment, it was necessary to make a baseline evaluation of the software development and test process. This was first done at the request of management to identify specific areas for implementing tools. However, the information gathered from the assessment also provides a

more generalized view of the development and test processes that is useful in understanding some of the team dynamics involved with tools, testing, metrics and automation.

TrackWear followed a traditional model of software development embellished with several practices borrowed from Agile methodologies. As would be expected in a traditional model, projects were executed in phases, from design to development to testing. But, there were also *scrum meetings*[3] twice a week to update progress and accommodate last minute changes. However, given the team's limited use of Agile practices (user stories, backlog, iterations, etc.) it was fair to characterize the process as traditional or "Waterfall".

Development efforts were organized as projects, usually consisting of loosely related functionality meant to be delivered in the space of two or three months. Project teams were small, often consisting of a developer-tester pair with additional support from an architect who worked on several projects simultaneously. User groups also participated on an ad-hoc basis to clarify requirements and evaluate the finished work.

Here are the phases of development as they are described in TrackWear's process documentation and as observed in the live environment.

Requirements/Design

This phase included the initial request for work based on customer need, integration or production requirements. It consisted of a series of informal discussions among users, developers, testers and management, captured in a text document that would later be referred to as a "design specification". These documents contained few detailed requirements or implementation details. Furthermore, they were understood to be only a placeholder for further discussion and refinement.

---

[3] The "scrum meetings" were an allusion to SCRUM, a well-known Agile development methodology.

Development

Once it was felt that the team had enough information to begin coding, developers would start implementing functionality. At the same time, testers would start writing new manual tests, or changing existing ones to align with the work of development.

Module Testing

As developers finished coding pieces of functionality, they would build the software on their desktop. Then, they would deploy it to a shared environment where testers could start working with the new code and refining their tests. At this point, the team might also present new functionality to users in the form of a prototype. Any issues or defects founds at this level of testing were captured informally, or simply discussed with the developer, rather than being entered into the defect tracking system.

It is important to note that software produced and tested during the module testing phase was understood to be far from complete. Therefore, in this phase testers were not trying to "validate" the software. Instead, they acted as "co-developers", partnering with those writing the code to understand functionality and identify problems. While this practice resembled the kind of iterative work done by developers and testers in an Agile environment, it was more informal and exploratory. This was due in part to the fact that module testing environments usually lacked significant components or integration points that would have been necessary for more authoritative testing.

Integration Testing

The integration test phase was the point at which the software was deployed to an integrated test environment. In this phase, functionality was presumed to be complete and had been tentatively approved by users and management. Testers would execute tests created for

new functionality, as well as any regression tests identified as relevant to the project. Defects were captured in the defect tracking tool. From this point on, the test team helped manage the project's release by estimating the time it would take to fix individual defects and to complete work on the project.

Release Certification

Integration testing ended when the tester assigned to the project certified that the project had fulfilled its requirements and that the software contains no serious defects. Interestingly, there appeared to be no structured approach to this certification, only the word of the tester.

Installation "Smoke" Testing

This was the final step in deploying new software to TrackWear's production environment. A team consisting of developers, testers, and database architects would deploy the software to the production. This would be done late at night to minimize the possible impact on customers. Once the software was deployed, testers would execute a series of tests to validate the deployment. Though it was described as a cursory validation, installation testing usually took several hours to complete and was often delayed by configuration issues.

Evaluation of Current Practices

In addition to identifying TrackWear's life cycle model and detailing some of its practices, this evaluation offered the following view as to some of the strengths and weaknesses of the test effort.

Strengths

- Testers had detailed knowledge of the system's functionality relative to the business domain.
- Requirements were captured in design documents.

- Users were involved early in the process.
- Written tests captured the detailed steps to validate functionality.
- Testing was performed in controlled phases.
- Testers worked closely with developers throughout the development and test process.
- Team members articulated a consistent shared vision of the project life cycle.

Weaknesses

- Tests were kept in spreadsheets with no clear organization. Testing required detailed knowledge of the test library.
- Heavy reliance on end-to-end, system-level tests
- Test data created in real time
- No test automation
- Few metrics produced or used

The test team's most valuable asset at the time of this study was a large collection of detailed manual tests. The tests had been carefully created and maintained over a two-year period and represented hundreds of hours of work. The spreadsheets were organized according to the project for which they were created. All documents were then stored on a company intranet which was also arranged by project. At the time of this study, there were more than twenty project pages, each with one or more test sets attached.

With no metadata or tools, testing was planned, estimated and executed based on the personal knowledge, memory, and opinion of the tester. According to management, the time estimate for testing depended on which tester was asked. Similarly, the progress of test execution was only visible to the tester.

This research found that while TrackWear's testers were highly motivated subject matter experts, they lacked many of the skills and tools usually associated with modern testing. For example, while the test team had created an impressive inventory of tests, the planning and execution of test cycles seemed to rely on the tester's intuition rather than systematic analysis. There was little done to prioritize tests or to partition test problems. This resulted in long test cycles that were difficult to manage or scale.

The test team also produced few metrics to provide feedback to management and developers regarding the state of testing or the quality of the product. Instead, the organization seemed to rely alternately on informal, intuition-based estimates or hard "certifications" by testers. This resulted in suspicion by management that they were not being given enough information, and defensiveness on the part of testers, who were reluctant to be held responsible for any undiscovered defects.

Most remarkably, TrackWear's test team had failed to leverage any kind of technology to create or manage test data. This meant that testers were often required to recreate similar test data dozens of times throughout the course of a test cycle. More importantly, test data was usually created in real time using live hardware devices, a process which could take hours or even days.

Recommendations

- Prioritize test cases based on risk
- Estimate execution time
- Refactor tests to remove redundancy
- Optimize test execution using tools
- Start capturing and reporting simple metrics.

- Develop a method for generating test data
- Automate the most important tests

Compuware Tools

Compuware's QACenter is set of suite of commercial tools designed to support a number of common SQA artifacts and activities. The suite features QADirector (QAD) for managing tests and TestPartner for creating test automation. It also includes tools for tracing requirements and manipulating test data. The tools can be configured to support an organization's existing test process.

QADirector 06.00

Enables central management of test assets such as scripts, test plans, test results and test requirements. Provides visibility into the test process with dashboard and reporting features. The product is a web application running against a SQL Server database. Users access a web page to install the client. All clients access the same repository.

TestPartner 06.02

Allows users to record or script automated tests. The tool provides also provides an environment for test execution and the collection of results. Tests can be executed on multiple remote machines simultaneously. TestPartner is a Windows application that uses ODBC to access a central SQL Server database.

TrackRecord 06.03

Bug tracking database with central repository. Integrates with QAD for traceability between defects and tests.

File-AID/CS 04.02

Enterprise test data management tool. Allows users to extract, load, convert, transform, generate, compare, and edit test data through a graphical interface.

Plan to Use Compuware Tools

The recommendations presented to management centered on three essential components; the optimization of manual testing, test automation, and metrics. It was thought that by removing redundancy between tests and prioritizing execution based on risk, SQA could reduce the time required for regression testing. In addition, test automation was put forth as a way to speed up execution of the most important tests and provide faster feedback. Finally, better metrics were needed to communicate with management, as well as to measure the effectiveness of any changes made to the process.

Summary

TrackWear has a traditional management structure. The organization had made several efforts to implement various tools to support the development and testing. TrackWear's software development lifecycle followed a traditional model with phased testing. Test process strengths included testers' detailed knowledge of the product and business domain, as well as a large collection of detailed manual test scripts. However, storing the scripts in loosely organized spreadsheets made it difficult to manage test suites or estimate the time needed for test cycles.

The Compuware tools already purchased by TrackWear included functionality designed to address some of the specific issues that had been raised by management and discovered in the course of this study: QAD to manage manual tests, test execution and metrics; FileAid for the creation of test data; TrackRecord to integrate defect tracking; and TestPartner for test automation.

Chapter 6: Pilot Project

The method chosen to implement the selected Compuware tools in TrackWear's test environment was a pilot project. The idea was to insure that "…that any problems encountered in [the tool's] use are ironed out when only a small number of people are using it" (Fewster & Graham, 1999, p. 290). In addition, the pilot project was intended to help evangelize the Compuware tools and generate interest in their use. Although management did not expect the team to spontaneously migrate their work to the new tools, the hope was that the pilot would help create a robust framework that the organization could adopt as time and resources permitted.

The project chosen as a pilot vehicle for the new framework was called "Federal Requirements" based on the fact that the functionality was being developed for a bid to be submitted to the federal government. Testing for this project would involve implementing the Compuware tools in three areas; managing manual tests and test execution, test automation, and the creation of metrics.

Scope

Managing tests for the Federal Requirements project would consist of capturing new manual tests using QAD, identifying the appropriate regression tests and importing them from the existing spreadsheets into the tool, then executing module and integration testing using the QAD test execution client.

TestPartner would be used to create a number of automated tests to demonstrate how the tool could be used to interact with the *software under test* (SUT) to validate functionality or automate test set-up. TestPartner would also be integrated with QAD so that automated tests could be managed from the same interface as manual tests, and have their results included with

those of manual tests. Finally, QAD would be configured to provide test execution metrics on manual as well as automated tests. These would include the number of tests planned, percentage of tests completed, and percentage of pass/fail.

The pilot project would not include the migration of regression tests to QAD beyond those identified as part of the Federal Requirements project. In addition, the metrics delivered with the project would be limited to the QAD metrics dashboard as it was configured more or less out of the box. The pilot would also not include any significant reconfiguration of the dashboard or creation of any custom reports.

Automated testing for the pilot would be limited to several scripts that would demonstrate some of TestPartner's functionality as well as its integration with QAD. These scripts were not expected to validate functionality that was not being validated manually. In addition, the pilot would not address a comprehensive strategy for automated testing.

Deliverables

In addition to testing the software to be delivered as part of the Federal Requirements project, the pilot would deliver the following:

- Manual tests for the pilot project in QAD
- Demonstration of how a completed test project is structured and executed using the tools
- Metrics Dashboard – presentation of "live" test metrics based on actual project artifacts
- Automation Demo – TestPartner automation scripts to show feasibility of the tool and possible strategies for implementation
- Test results from the Federal Requirements project indicating that the software was ready for release.

Timeline

Development on the Federal Requirements project had started in mid-March 2009 and was about half finished when the decision was made to use the project as a pilot for the emerging test framework. Figure 1 shows the activities related to project testing and implementation of the tools.

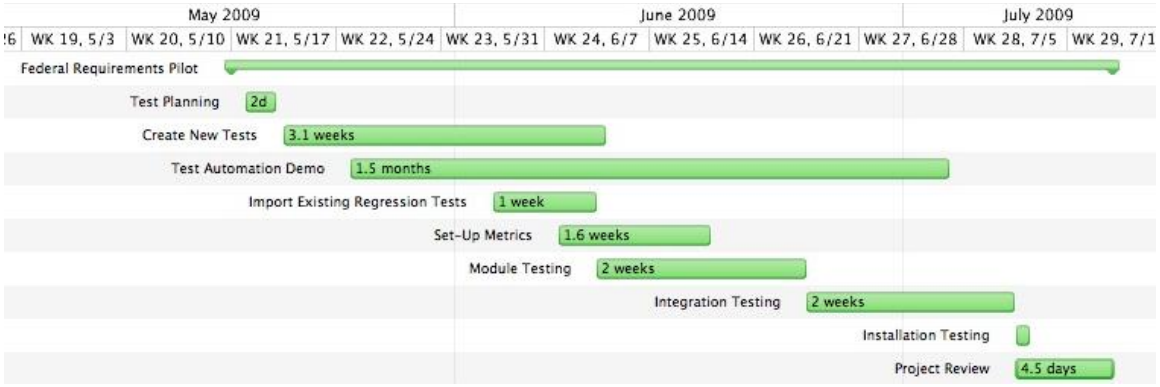


Figure 1 - Pilot Project Timeline

Activities

Final development and testing for the pilot project took place over a two month period in the summer of 2009. The following is a general description of the activities shown in Figure 1.

Test Planning

When the decision was made to test Federal Requirements with the new framework, the planning and design documents for the project were reviewed and a test plan was created. The plan was based on a simplified version of the test plan template outlined in *Systematic Software Testing*. (Craig & Jaskiel, 2002, p. 62) Once the test plan was approved, the activities and schedule were merged with the project plan so that test and development activities could be viewed simultaneously.

Creating New Tests

All of the new tests created for the pilot project were scripted in the QAD interface. The process of creating new tests was essentially the same process that was already being used at TrackWear. Here, the tester works with design documents and collaborates with the developer to understand functionality and capture it in test scripts.

Test Automation Demo

Automated testing had been a major goal driving the purchase of the Compuware tools, as well as the effort to configure and use them. Stakeholders often mentioned automated testing as an area where they saw great potential. However, some of this enthusiasm seemed to be based on the expectation that automation would replace manual tests. However, most of the existing tests involved physical interaction with hardware devices, tests which Fewster and Graham (1999) suggest should not be automated at all. (p. 23)

The automated scripts created for the pilot project were all designed to take data from a spreadsheet and enter it into the system through the user interface. Each script would log in to the program and create users, administrators, and other entities related to the business model. The scripts were designed to continue entering data until they reached the end of the data file. This meant that a tester could run one script to enter dozens or even hundreds of lines of data. Furthermore, these prototype scripts were modular and could be recycled to handle different data from different entities.

This strategy had the advantage of demonstrating several important concepts. The scripts that were created showed how well TestPartner interacted with the SUT. They also showed how data repositories could be used to drive automation. Most importantly, integrating the scripts

with QAD showed how automated tests could be run in the same context as manual tests and provide the same kinds of metrics.

Importing Regression Tests

A critical step in moving toward a more systematic approach was to start moving tests into a format in which they could be viewed, compared and edited by multiple users. This would help the team to begin factoring out redundancy by allowing them to view the tests as a complete inventory. This view makes it easier to identify duplicate, outdated, and redundant tests. Organizing the test inventory into functional areas would also provide the context for assigning risk as well as time estimates to test sets. QAD provides a graphical interface with a hierarchical structure in a shared environment to enable these activities.

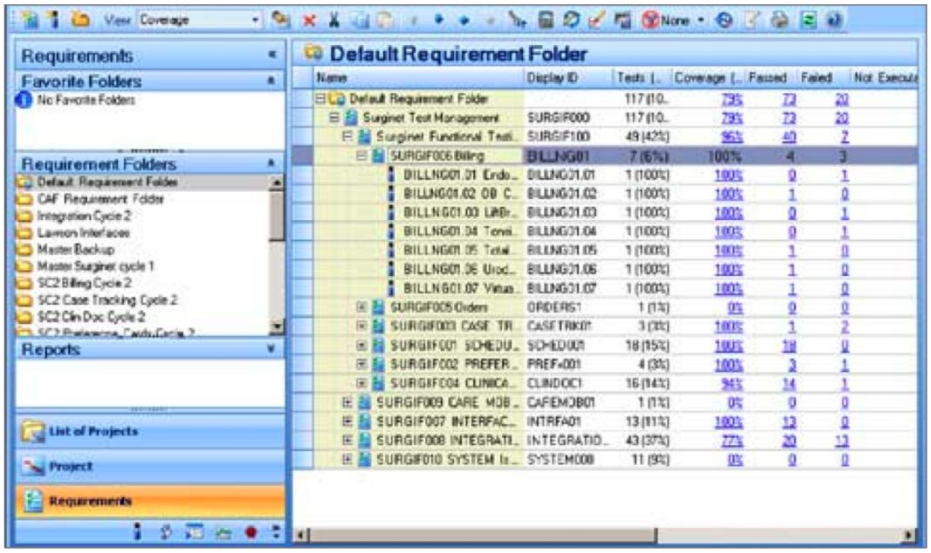


Figure 2 - Requirements in QADirector

The regression tests that had been identified as critical to the project were imported from the existing Xcel sheets into QAD. Once imported, each test was assigned initial estimates for execution time and risk. These estimates were understood to be rough, but they demonstrated how risk and execution time can be used to organize a test cycle.

One challenge in adapting TrackWear's test inventory to the new tool was that QAD is a requirements-based tool designed to enable traceability and calculate test coverage. Therefore, linking tests to requirements was key to much of the tool's functionality. Since TrackWear's SDLC did not produce requirements in this form, a solution was needed to fill the gap.

This problem was solved by reverse engineering requirements based on the tests themselves. The meticulous construction and maintenance of the manual tests meant that the tests themselves could be said to represent the requirements of the product. Therefore, it was possible to create requirements in QAD by examining each test and writing a "requirement" based on the functionality being tested.

Metrics

QAD's dashboard provides automatic tracking of a number of critical test metrics, enabling real-time visibility into the test process: What percentage of tests have been executed? How many have passed? What areas are producing defects? Managers, developers and test leads can find the answers to these questions by simply logging in to . However, in order for these metrics to be meaningful, the underlying data must conform to the model. The process of knitting together the data and configuring the tool is far from trivial.

This part of the pilot highlighted two recurring issues with the Compuware tools: the extent to which TrackWear's process would need to change to accommodate the new model, and the need for a dedicated resource to administer QAD.

Module Testing

Module Testing at TrackWear consisted of testing smaller pieces of functionality in a non-integrated environment. This phase of testing uses various stubs to replicate integrated functionality not yet present in the environment. TrackWear's model for Module Testing turned

out to be a much more informal and collaborative process than it first appeared. While the expectation was to see a controlled deployment of more-or-less finished software, in TrackWear's version of Module Testing, testers and developers were still defining requirements. Therefore, much of this phase of testing consisted largely of test creation rather than test execution.

Integration Testing

In preparation for Integration Testing, all of the tests for the pilot that had been captured in QAD were organized into execution plans. The execution plan was another key artifact in the Compuware model that contained not only the tests to be executed, but the total time required for execution, as well as the tester assigned to run the tests. Once a test set had been assigned, a tester would log in to their QAD client and their test assignment(s) would appear in a pop-up window. Then, the client would guide the execution of the tests, allowing the tester to enter the results. The results would then be populated in the QAD database enabling the team to track test progress and results in real time through the dashboard.

While this functionality seemed to address management's concern over the visibility of the test process, it also required a fair amount of adaptation of the existing process. Creating requirements and execution plans, assigning risk and time estimates, and using the execution tool to run tests—all took time and effort. Moving beyond the pilot project, the challenge would be convincing the team that the results justified the extra work.

Installation Testing

At TrackWear, installation testing consisted of deploying the finished software to the production environment and running a pre-determined set of tests to validate the configuration. While testing in a production environment carries the risk of corrupting live data

or breaking functionality, TrackWear insisted that it was necessary to identify problems with the complete system.

Project Review

Review of the project involved several meetings with developers and management after the software had been successfully deployed.

Chapter 7: Results

The pilot project demonstrated the feasibility of integrating the Compuware tools with TrackWear's SDLC to provide test management, automation and metrics. TrackWear now had a technical framework to enable changes in these areas. In addition, the project had shown how the company's processes and artifacts would need to adapt in order to accommodate the new model. The project had also given a sense of how much ongoing effort would be required to maintain the tools and processes.

But, it was clear that more work would be required. Because, while the pilot had certainly demonstrated the *technical* processes involved with test management, automation and metrics, the institution had yet to fully develop the *conceptual* underpinnings of such a framework. This is illustrated by some of the comments received from various stakeholders, during and at the close of the pilot project, regarding the following issues.

Test Management

The testers had been fairly outspoken about their distaste for QAD. They saw no reason to change the way tests were managed or executed, and said the extra time and effort needed to use the tool could be better spent testing the product. When the issue was framed as a new requirement from management, they argued that management's expectations were unrealistic.

Furthermore, it was said that management often made demands one day and forgot them the next. This perception seemed to point to issues beyond the scope of this project.

This attitude was not limited to the test team. Developers, for their part, were largely neutral with regards to how tests were managed and instead seemed anxious to defend the test team's methods by pointing out the demanding schedule and lack of resources.

On the other hand, the Pilot had demonstrated the relative ease with which the existing test inventory could be imported into the tool. This ease of migration represented a positive impact on the value proposition for using QAD to manage tests.

Automation

There was a lot of enthusiasm about the test automation created for the pilot project. Testers had complained about the fact that they had to go into the application and create data each time they wanted to run a test. The data-driven scripts created for the pilot were seen as a welcome shortcut that would relieve testers of some of their more tedious chores. Though there still seemed to be little understanding of the role test automation might play beyond that of emulating manual tests, this was a success the team could build on.

Metrics

The pilot project demonstrated that once the various artifacts and processes are configured correctly in QAD, metrics can be gathered, published, and updated automatically. However, the team still had to develop the kinds of processes and decision points that would make the metrics useful. For example, after a demonstration of the metrics dashboard, one manager remarked "I just want to know when we'll be done!" While the metrics produced for the pilot would certainly help make that kind of projection, it would take practice, patience and cooperation.

Summary

The goal of the pilot was to implement a newly created test framework on a real project. This study has focused on three major areas the client had hoped to improve through the use of tools: test management, automated testing, and metrics. The pilot project followed a written plan that included clear definitions of goals, scope and responsibilities. Execution of the plan centered on a project selected by management as the vehicle for the pilot. Evaluation of the pilot's success was based on its having achieved specific goals, as well as on feedback received from management and team members.

While the project met its technical goals and produced some valuable information, feedback from stakeholders was mixed. There was a lack of progress in socializing some of the concepts behind the project. While the team may have learned how to organize tasks and produce metrics, they were not yet convinced that these activities would be useful in solving their particular set of problems.

One way this manifested itself was in team member's resistance to some of the changes required to implement the new framework. It was often difficult to establish consensus about a process when team members were unsure of its value. In addition, there was reluctance to put effort into "process improvement" because of the perception that management could change course.

Perhaps the greatest area of resistance was against the fundamental truth that improving the development or test process requires ongoing participation by multiple stakeholders. Some were of the opinion that if the right person could implement the right tool, the whole team would benefit.

Chapter 8: Conclusions

This study follows an organization's second effort to implement a set of SQA tools. Like many companies, TrackWear had discovered that buying the tools was not enough to solve their problems. Without sustained effort, the Compuware tools had been well on their way to becoming shelfware. Meanwhile, lacking adequate tools and processes, SQA had continued to fall behind in its ability to serve customers. This led to an intervention by management that prompted the work on which this study is based.

Tools First

The fact that this study focused on tool implementation does not necessarily mean that tools were TrackWear's most important problem. Instead, this study found that management's primary concerns about the test process, namely that testing took too long and was difficult to track, had more to do with test strategy than tool implementation. The more fundamental problem seemed to be an over reliance on manual, black-box testing. This issue would not be solved by simply using tools to manage the existing process.

This finding seems to support the body of knowledge that suggests putting the development process before tools. However, this study found there was still some value in the tools-first approach taken by the project. First, the effort of implementing the tools facilitated a detailed discussion of the test process. Second, it helped introduce a number of core concepts that would help the test effort regardless of what tools were used. Finally, focusing on the tools seemed to reduce the perception that the testers were being unfairly criticized for problems with the development process.

However, building a process around a pre-selected tool also carries the risk that the results will match the tool, but not the needs of the organization. This study found that once

implemented in TrackWear's environment, some things fit and others did not. So, while the added functionality of the Compuware tools offered some near term improvements, the tools did not appear to offer the kind of end-to-end solution envisioned by management.

Practical Uses

While the integrated functionality of the Compuware tools would not solve all of TrackWear's test problems, the study found several specific areas in which the tools could be used to make incremental improvements. For example, this study finds the TestPartner automation tool to be well suited to the technology and skill level of TrackWear's testers. Given the necessary resources, testers could easily start using automation to speed up testing or data configuration.

FileAid, the data creation and extraction tool, is another component that offered some interesting solutions in terms of setting up test data. However, using the tool effectively would require additional research and collaboration. In addition, the steep learning curve was a limiting factor.

Finally, managing manual tests with QAD seemed to be a viable point of entry to better test design and execution, even though the team would probably not use all of its functionality. The ability to create, edit, and execute tests from a shared repository represented a major improvement over the existing process. While this might not lead to immediate optimization, it could facilitate better testing by helping the team visualize and measure the process.

Value Proposition

Tool implementation depends on an organization's ability to establish and maintain a value proposition relative to the tools. If the perceived benefits of the tools had failed to outweigh the

perceived costs, testers would be not have been encouraged to adopt them. Taken as a whole, the project had mixed success "selling" the tools inside the organization.

On one hand, the automated scripts were an instant success. In fact, the study found that the high perceived value of test automation helped balance the value proposition for the whole project. However, the negative perception of the Compuware tools' usability may have caused sufficient controversy to tip the balance in the other direction. The prevailing attitude among testers was that the tools were needlessly complex and did not return sufficient value for the effort required to learn and use them.

Finally, the study found that the decision to hire a contractor to help with the Compuware tools was a critical step toward establishing a value proposition for implementation. Based on testers' workload and the amount of effort required to make the tools functional, it would have been unreasonable to expect the test team to set them up "in their spare time". The extra resource was also intended to demonstrate management's commitment to the project, another important factor in developing the value proposition.

Ultimately, the study found that establishing the value of the tools with TrackWear's test and development team was hindered by the fact that much of the value being offered was theoretical. In addition, the proposed changes were based on concepts which represented a systematic, context-driven approach to software testing. To the extent that TrackWear had neither developed nor socialized such an approach, selling the program was more difficult.

Further Research

Real Value

The results of the study seem to beg the question: If SQA tools in general or automated testing in particular has the potential to radically improve software delivery and quality, why are

so few organizations using them effectively? Or, in this case, why was it so hard to even establish their theoretical value?

One answer is a lack of institutional knowledge about software testing. Despite all evidence to the contrary[4], many remain skeptical that a scientific approach to testing will produce better results than an unscientific approach. The fact is, organizations like TrackWear manage to produce software, make a profit, and sustain their business model year after year, relying on informal methods and untrained testers to validate software quality. This leads to the conclusion that either the testing being performed is in fact "good enough" or that the level of quality required for success is low.

This apparent contradiction represents the single greatest challenge to improved testing, and may threaten the continued development of SQA as a discipline. Further research is needed to investigate the costs and opportunities of testing, and grow the value proposition of improved SQA.

Feelings

This study has focused in part on how implementing new tools involves presenting a technical and conceptual view of testing. However, as Kaner et al. (2002) point out, "Sensibility is not the issue; feelings are. No matter what else it's about, process improvement is always about feelings" (p. 9).

TrackWear's test process required its testers to execute many long and tedious tests, often after-hours and on weekends. While this was a source of delay and frustration to the organization, it was paradoxically a point of pride and respect on the part of testers and their

[4] The National Institute for Standards and Technology estimates that inadequate software testing costs businesses and consumers almost $60 billion a year. (NIST, 2002)

colleagues. Testers were often described as "tireless" or "committed" because of their willingness to put in overtime testing the product.

If the goal of improving the test process is to limit the team's reliance on tester stamina, how would that affect the way testers and their colleagues saw the value of their work? It bears investigating whether this may have represented a fundamental issue with regards to changing the test process.

Gatekeeper Role

One issue that was mentioned in the description of TrackWear's test process was "release certification". Kaner et al. (2002) warn testers "Never be the Gatekeeper!" pointing out that "…when testers control the release, they must also bear the full responsibility for the quality of the product" (p. 8). It is unclear how much influence this "gatekeeper" role may have had on how testers viewed the value of change in general, or the Compuware tools in particular.

Summary

TrackWear's implementation strategy, though arguably flawed, represents a common way of implementing SQA tools. Despite evidence to the contrary, many organizations see this kind of tools-first, top-down strategy as the best way to get tools running in their environment. This study takes this strategy at face value and traces its outcomes in terms of tool adoption and improved testing.

While the study finds some value in using tools as a way to analyze and discuss the test process, it also points out the danger of implementing a tool that fails to serve the needs of customers.

The work described in this study has enabled TrackWear's test team and management to develop a number of new technical capabilities. However, continued progress will depend on

their ability to grow the value proposition around a systematic approach to software testing that includes risk analysis, test automation, and metrics.

Finally, this study demonstrates the need for further research to understand the true cost of quality and how it relates to the success of the organization and the future of SQA.

References

Black, R. (2002). Managing the Test Process: Practical Tools and Techniques for Managing Hardware and Software Testing. New York: Wiley Publishing, Inc.

Craig, R., & Jaskiel, S. (2002). Systematic Software Testing. Boston, MA: STQE Publishing.

Fewster, M., & Graham, D. (1999). Software Test Automation: Effective use of test execution tools. London: Addison-Wesley.

Hutcheson, M. (2003). Software Testing Fundamentals: Methods and Metrics. New York: Wiley & Sons.

Ince, D. (1987). The Automatic Generation of Test Data. The Computer Journal , 30 (1), 63-69.

Kaner, C., Bach, J., & Pettichord, B. (2002). Lessons Learned in Software Testing: A Context-Driven Approach. New York: Wiley & Sons.

Kaner, C., Falk, J., & Nguyen, H. (1999). Testing Computer Software. New York: Wiley & Sons.

Kohl, J. (2007 йил 01-12). Man and Machine. Retrieved 2009 йил 22-08 from Better Software Magazine: http://www.stickyminds.com/BetterSoftware/magazine.asp?fn=cifea&id=103

Lewis, W. (2005). Software Testing and Continuous Quality Improvement. Boca Raton, FL: CRC Press.

National Institute for Standards and Technology. (2002). The Economic Impacts of Inadequate Infrastructure for Software Testing. Gaithersburg, MD: NIST.

Pan, J. (1999 йил Spring). Software Testing. Retrieved 2009 йил 17-08 from Carnegie Melon University: http://www.ece.cmu.edu/~koopman/des_s99/sw_testing/

Patton, R. (2006). Software Testing. Indianapolis, IN: Sams Publishing.

Scarpino, J., & Kovacs, P. (2009). An Analysis of Software Quality Assurance Tool's Implementation: A Case Study. Issues in Information Systems , IX (2), 146 – 152.

Spillner, A., Linz, T., & Schaefer, H. (2007). Software Testing Foundations. Santa Barbara, CA: Rocky Nook, Inc.