Challenging Conventional Wisdom

Abstract

The earth is flat. Mankind will never fly. Reasonable people believed these "facts" for thousands of years, but advances in knowledge and technology proved them wrong.

Does the software testing industry have any such "facts"? In this paper, I will explore common testing beliefs. Using experiences from SAS, I will attempt to confirm or refute these pieces of conventional wisdom.

Introduction

If we all worked on the assumption that what is accepted as true is really true, there would be little hope of advance.
Orville Wright

SAS has a long-held commitment to delivering quality products. R&D hired its first tester nearly twenty years ago. The company then hired the first member of its QA group soon thereafter. R&D continued with two types of testing groups until earlier this year. QA existed under a separate management structure as an independent testing group. Most R&D divisions created their own R&D testing departments and located them (both physically and organizationally) near their software developers.

In February 2001, the testers from QA moved into the testing departments of the R&D divisions. In addition, a small, centralized organization was created to provide global testing services. Since then, SAS testers have grappled as a community to reconcile the processes and mindsets that have governed our past---and that will shape our future.

In the philosophical discussions that followed the reorganization, I heard many varying opinions about what testers do and how they should do it. I was prepared for and expected to hear different perspectives from the former QA staff. But I found myself surprised by the diverse opinions among the existing R&D testing groups.

Sometimes it was hard to tell whether these passionately-held beliefs were fact or fiction. I decided to investigate four of the more frequently repeated statements---to see whether SAS had any experiences to support these opinions. Evaluated in this paper are:

- Requirements are a requirement.
- Test plans improve testing results.
- Working closely with developers pollutes a tester's mind.
- Testers get no respect.

The intent of this paper is to evaluate whether our beliefs hold up in practice. I used two main sources of information. Our DEFECTS tracking system provided the objective data. To collect subjective data, I interviewed and surveyed the testers, developers, and managers in R&D.

Requirements are a requirement.

A pessimist sees the difficulty in every opportunity;
an optimist sees the opportunity in every difficulty.
Winston Churchill

Years of research have shown how important requirements are in developing top quality software. “A complete understanding of software requirements is essential to the success of a software development effort. No matter how well designed or well coded, a poorly analyzed program will disappoint the user and bring grief to the developer.” (Pressman, 1992).

Accepting these points, I explored how the absence of requirements impacts testing. I wanted to gauge whether testing efforts are seriously harmed if a tester receives a project without the appropriate requirements specifications. In other words, can good testing happen without requirements?

Understanding that many factors affect software quality, I searched for a case study with limited variables. I found a pair of R&D projects that were appropriate for the study. The pair of projects was similar in many ways but had two major differences separating them: (1) the second was more complex than the first; and (2) the first was handed off to the tester without requirements.

The same developer wrote the code for both projects. Both projects were based on industry standards. The developer borrowed much of the second project's core functionality from the first project. Although the same developer wrote specifications for both projects, the tester received specifications for the second project only. I was told that the tester "didn't ask for them the first time around."

The same tester tested the two projects. For the first project, "Project NoReq", she received one sample test and a rough draft of a user reference guide. She had no knowledge of the targeted market, the goals of the project, or the problem it hoped to solve. She had two months to test Project NoReq and spent a frustrating two weeks of it getting up to speed. Once this was done, she enjoyed working on the project. Project NoReq was her primary focus during that period but she did support one other project as well.

If there had been a famine of information with Project NoReq, there was a feast with the second, "Project Req." The tester received an edited user reference guide, two sample tests, functional specifications and the requirements for the industry standard upon which the project had been based. Just as the developer had borrowed code for Project Req, the tester adapted test cases from the earlier project. She spent two days getting up to speed. She had two months to test Project Req; it was her primary responsibility but she did support several other minor projects at the same time.

During Project NoReq's testing cycle, the tester reported 1.4 defects/KLOC (1000 lines of code). During Project Req's testing cycle, she reported 2.0 defects/KLOC. Defect detection followed the same pattern for both projects: all but one defect were found in the first month.

To date, no external customers have reported defects against either project. Project NoReq has been in the field since Fall 1999 and Project Req has been in the field since Spring 2000.

The tester found the results astonishing. Each defect in the first project took so much effort that it seemed as if she had reported more. She wasted nearly two weeks thrashing about at the beginning of Project NoReq's testing period. She primarily used "ad-hoc" testing---which was a fun way to test. It took about a week to convert her “ad-hoc” test cases into an automated regression suite---which was not fun.

The defects for Project Req came more easily. The higher complexity in Project Req was somewhat ameliorated by her experience with Project NoReq and the regression suite she adapted. The tester used methodical techniques with Project Req---which was a more efficient and, in this case more tedious, way to test. With the same amount of time, she realized a 30% improvement in defects/KLOC---while juggling extra workload.

Are requirements necessary? With requirements, the tester has the potential for excellent testing. Without requirements, the ceiling on effectiveness is set lower. The testing effort can still be good, but the tester will report fewer defects and may need dedicated time to find them. The presence or absence of requirements sets the threshold on the effectiveness of the testing effort.

Test plans improve testing results.

A good plan, violently executed today,
is better than a perfect plan tomorrow.
George S. Patton

After our two testing groups were merged, the unified testing departments assumed the responsibility for writing and maintaining formal test plans. Most R&D testing groups had been preparing informal devices, such as test matrices or test inventories, for years. But few had extensive experience with the type of formal test plans patterned after IEEE standards.

A debate ensued about the goals of test plans. Everyone agreed that test plans created a good historical record of a software release, provided training documentation for new testers, and met expectations of our regulated external customers. But did formal test plans improve testing directly? Or, stated another way, did lack of test plans compromise testing results?

To answer the question, I looked first at one simple metric. Do testers who forego formal test plans report fewer defects than testers who use formal test plans?

To set up my case study, I considered recent development projects in our flagship product line. Each project should have a pair of test teams assigned to it; a team within the R&D department and a team within the independent testing group. I looked for a variety of testing styles---such as manual versus automated. I wanted to find test team pairs that maintained attitudes of tolerance or cooperation.

I finally settled on four development projects. Three of the test team pairs had the same size; in the fourth pair, the R&D test team had one more member. I measured the span of time the teams spent entering defects on each project. On average, the independent test teams used a year (four quarters) and the R&D test teams used slightly more (five quarters). Other than that, the test team pairs should have been relatively well matched.

The independent test teams coexisted within the same management structure. They were generally located apart from the developers but had access to project information through web sites, phone calls, and meetings. They used global automation tools provided from a global tools group. In addition, the independent teams had a dedicated support group to provide systems administration and tools. One set of these tools gave the independent teams test plan management. Test plans were required. Developers did not usually review these test plans.

The R&D test teams worked within the R&D divisions and were led by testing managers. They were generally located in close proximity to the developers they supported. They had access to project information through visits, web sites, phone calls, and meetings. The R&D test teams

also used the global automation tools. They had to fend for themselves for customized tools. Test plans were optional and, in fact, were not used by any of the R&D test teams in this study. Most used informal testing devices such as test matrices.

I started by studying the total number of defects entered on each project. In particular, I looked at the percentage of defects entered by both test teams: the independent teams (Test Plans required); and the R&D test teams (Test Plans optional). I also included information from one more group, our Technical Support Division, to gain a sense for how well the projects performed after release. The results are summarized below.

Project	Testing Style (Manual vs. Automated)	% of Total Defects: Test Plans Required	% of Total Defects: Test Plans Optional	% of Total Defects: Technical Support	Product Maturity
R	Mostly manual	29%	30%	<1%	1st release
S	Mix	5%	21%	4%	2nd release
T	Mix	5%	54%	10%	Mature
U	Mostly automated	7%	53%	<1%	2nd release

In every case, the teams without test plans found more defects than the teams with formal test plans. Even when I adjusted for the *headstart* the R&D test teams had, the gaps barely narrowed. For these four R&D test teams, lack of test plans did not compromise their testing results.

Could formal test plans have value that defects counts would not reveal? I posed the question to testers from a cross-section of backgrounds. They offered the following answers.

- Three testers believed that test plans had value in providing a format for organizing their testing materials. However, two of those testers found the official format “cumbersome” or “bureaucratic.” Both had devised their own informal materials—to which they referred from the formal test plans.
- One tester theorized that any fault with formal test plans lay not in the concept but in the execution. Lack of developer review was problematic.
- Many testers view testing as an evolutionary process or a craft. A detailed roadmap at the outset of a project would prove counterproductive for skilled testers.
- None of the testers believed that formal test plans improved their effectiveness when testing a specific project.

Do formal test plans improve testing? The objective data and tester interviews suggest otherwise. Formal test plans do not improve testing results directly.

Working closely with developers pollutes a tester's mind.

There is nothing either good or bad,
but thinking makes it so.
William Shakespeare

There appears to be a wide-body of support for close collaboration between developers and testers. “My best experiences with test organizations have been with … the project testing specialist and the software test support function. In each case dedicated testing is applied to the entire project---from start to finish---and in each case the testing resources work in direct support

of the development resources. This fosters close cooperation and avoids unnecessary redundant testing." (Hetzel, 1988).

In many companies, management makes a choice between centralized and decentralized testing organizations. Since our company recently merged all of its testing resources into R&D divisions, we are particularly sensitive to this issue. Testers accustomed to the centralized structure have worried that their independence of thought can be adversely affected by day-to-day interactions with developers. Are their worries justified? Testers who have spent their entire career in the decentralized groups are enthusiastic about working with developers. But could that be because they have not experienced the other structure? Is there any way to measure the differences between the two philosophies?

To discern the influence developers have on testers' effectiveness, I studied four individuals who had voluntarily transferred from the independent testing group into an R&D testing group years prior to the testing community merger. Each cited the desire to work more closely with developers as one of their reasons. So, in all instances, the testers were motivated to make the new mindset work.

It was going to be difficult to narrow down the variables. Three of the testers had changed products when they changed jobs. Three accepted newly-created positions. Two worked on brand-new products. It might be hard to compare their *before behavior* with their *after behavior*.

But I gave it a try. I checked several objective factors: defect counts; defect detection pattern; defect severity; and the final status of the defects. The results were intriguing.

Defect Counts

Every tester had a decline in the number of defects entered after the transfer, including the tester who had not changed products. I admit to being taken aback by these results.

Tester	% Change in Defects Entered
W	-20%
X	-55%
Y	-9%
Z	-86%

Defect Detection Pattern

After moving into the new group, did the testers tend to enter defects earlier in the releases? The answer was "not really." There were no consistent patterns for three of the testers; defects were clustered at the beginning, middle, and end of releases---both before and after their transfers. One tester had a steady pattern for all releases, in either job.

Severity of Defects

A slight trend popped up. We have four severity codes for DEFECTS-- ALERT, HIGH, MEDIUM, and LOW—with ALERT being the most severe. All but one tester experienced a decline in the number of defects they entered at the two highest severity levels.

Tester	% Change in Defects Marked ALERT/HIGH
W	-14%
X	-27%
Y	-4%
Z	+7%

Final Status of Defects

I saw interesting changes in three of the status codes we track for defects. Those codes were: FIXED (the defect was fixed); DUP (the defect was a duplicate of another entry); and NOBUG (the defect was not really a bug). The results are summarized below.

Tester	%Change in FIXED	% Change in DUP	% Change in NOBUG
W	+17%	<-1%	-6%
X	+21%	-11%	-7%
Y	+11%	-4%	-1%
Z	+19%	-15%	-6%

After studying the objective data, I could draw a few conclusions from the testers' *after* behavior.

- They entered fewer defects.
- The defects they reported tended to be less severe.
- They were more likely to enter fixable defects.
- They were less likely to report duplicate defects or non-issues, an efficiency gain for developers.

The data were encouraging but I needed to know more. The declines in number and severity of defects were not intuitive. So I visited these testers, separately, to ask. Were these results predictable? And did they believe they were more or less effective after the move? All four testers contributed opinions, which I have condensed in the next section.

The testers were not surprised that defect counts decreased.

- I had collected the number of defects **reported**--not the number of defects found. *Before the move,* they reported 100% of the defects they found. *After the move,* they reported only a fraction.
- The before-to-after paradigm shift took them six to twelve months. The two years before they transferred were their most productive in the independent test group. The two years after they transferred were their least productive in the R&D test group---as they struggled to learn new technology, master new processes, and build closer relationships with developers. I had compared their best *before* years with their worst *after* years.
- They found defects earlier in the cycle; therefore, they were more likely to report them via e-mail or visits. In some instances, developers would enter the defects found by testers.
- Within the R&D test group, there was no implication that a tester's performance was related to the number of defects they reported.

The change in severity was understandable.

- The testers had changed products, which meant they had changed developers. The new developers were better.
- They found the serious defects earlier in the release, when they were less likely to be entering all defects in the tracking system.

The testers would have expected their FIXED percentages to increase.

- R&D testers often discussed problems with developers prior to reporting defects. Many defects were 'pre-approved' for fixing.

- The testers were able to devote more time to learning the product. They really did have a better sense of what was worth fixing.
- With better understanding of the product, the testers wrote better defect entries. Developers could investigate and resolve problems more easily.
- The testers were less likely to enter 'debatable' defects. They had more time to research problems. Being familiar with the pressures faced by developers, testers were motivated to narrow down or eliminate non-issues.

The testers believe that they have become more effective testers since their transfers.

- They prefer working more closely with developers.
- Testing projects from “start to finish” benefits the project and the tester.

Can testers lose their independence of thought by working closely with developers? As one tester acknowledged, the “potential for pollution” does exist. Being aware of the possibility helps to prevent it. The negatives of close relationships are overshadowed by the positives: shaping the product through early feedback; feeling part of the team; and gaining a deeper understanding of the product and the context in which it functions. It is a tradeoff but is well worth the risk.

Testers get no respect.

I do not want people to be agreeable,
as it saves me that trouble of liking them.
Jane Austen

“Developers don't respect testers.” “Testers are second-class citizens.” “Developers wish testers would disappear.” I have heard variations on this theme throughout my testing career.

While I admit to being on the receiving end of poor behavior from developers, it has been the exception, not the rule. I reviewed the results of a survey conducted at SAS in 1998 that addressed the working relationships between developers and testers. It showed that testers typically view the relationships more negatively than developers. However, since the survey did not use probability sampling and also had a low response rate, the results were not scientific.

With the assistance of a statistician, I conducted another survey using probability sampling. The survey population was defined as non-managerial developers and non-managerial testers who worked on products destined for external customers. I excluded R&D managers and testing managers to avoid opinions that may have been influenced by employee feedback. I surveyed only those R&D groups working on products for external customers since their testing organizations tend to be mature and share common structure and methodologies.

Using the survey population criteria, I constructed sample frames of 443 developers and 175 testers. From these, I selected two simple random samples. The sample sizes were determined

upon achieving sufficient precision and then adjusted upward to compensate for expected nonresponse. As shown below, the final sample sizes were 80 developers and 57 testers.

	Sample Frame	Sample Size	Number of Respondents	Response Rate
Developers	443	105	80	76%
Testers	175	71	57	80%

A separate web-based questionnaire was administered to each sample group of developers and testers. To help minimize nonresponse, each questionnaire was limited to just four questions with room for optional comments. (See Appendix I for the survey questions.) The questions posed to the developers and the questions posed to the testers addressed the same subjects but were put in terms of their respective roles in the relationship.

Two of the questions focused on the perceived respect developers have towards testers. Because there are many ways to interpret what "respect" actually means, I reduced the ambiguity by narrowing it down to two aspects. First, do developers and testers share positive working relationships? And, second, do developers value the contributions made by testers? The next two tables contain the percentage estimates for the different response categories of the two questions addressing respect. The standard errors of the percentage estimates are given in parentheses.

Responses to the Question Concerning Working Relationships

Response	Developers: how many testers do you have a positive working relationship with?	Testers: how many developers do you have a positive working relationship with?
All	81% (3%)	63% (5%)
Most	14% (3%)	32% (5%)
Some	4% (2%)	5% (2%)
None	1% (1%)	-

Responses To the Question Concerning Tester Contributions

Response	Developers: how often have you valued tester contribution to the development process?	Testers: how often do you think developers value your contribution to the development process?
Almost always	65% (4%)	26% (5%)
Most of the time	30% (4%)	58% (5%)
Some of the time	4% (2%)	16% (4%)
Never	1% (1%)	-

From the optional comments, I found some clues to the gaps in perception. Developers were generally quite positive and complimentary in their comments but a few voiced the following complaints:

- Testers need to be more proactive.
- Testers display a lack of interest in testing new features.
- The attitudes and training of some testers make them a bother and a chore to deal with.

Most comments from testers were positive as well. Yet they, too, offered hints to the differences in perception.

- Poor communication exists between developers and testers.
- How testers conduct themselves impacts their relationships with developers.

Do developers respect testers? This survey confirmed the impressions of the previous survey. Developers tend to respond more favorably toward testers than testers perceive. Developers are more likely to report positive working relationships with **all** testers than testers report with **all** developers. Likewise, developers value tester contributions **almost always** at much higher rates than testers think they do. Developers at SAS respect their testing staff.

Conclusions

The earth is round…from space. Mankind can fly…with help. What once seemed impossible is now true, depending on one's perspective. As I studied the points in this paper, I found the same phenomenon. I was able to draw conclusions but they were always with qualifications.

Requirements are a requirement. I was able to demonstrate a significant improvement in one tester's ability to detect defects by adding access to requirements. This increase seems even more remarkable given the tester absorbed a bigger workload during the period. Receiving requirements meant better effectiveness and efficiency.

However, the testing effort without requirements was not a poor one. If undiscovered defects lurk in the product, customers have not reported any yet**.**

Insight: I do not think I have ever met a testing manager, at SAS or elsewhere, who believed they had sufficient staff to get the job done. I know many R&D managers who complain about the lack of testing resources. Delivering requirements to testers could be an instant resource gain.

Formal test plans do not improve testing results. I reviewed objective data concerning four projects released within the last four years from SAS. I tried to select the projects carefully so that the variables affecting the testing efforts were as limited as possible. In all cases, the groups who did not use formal test plans reported more defects than those who did.

Insight: I firmly believe that formal test plans have value. We must be able to describe what we have done and how we did it---for both internal and external audiences. However, we should acknowledge them for what they are---important historical documents that take time and resources to produce and maintain. Then, we should give our testers the flexibility to create testing devices that meet the specific needs of the projects they support.

Working closely with developers does not pollute a tester's mind. When testers move closer to developers, do they lose their independence of thought? Slicing-and-dicing defects data was interesting but inconclusive regarding this statement. I queried the testers about their experiences and shared their comments. They prefer close relationships with developers and believe their effectiveness has improved.

Insight: If they allow it, testers can be influenced negatively by developers. But working closely with developers *doesn't have to* pollute a tester's mind. With open minds and hard work, testers can reap the benefits of teamwork without sacrificing their effectiveness.

Testers do get respect. 95% of both developers and testers claimed that *all* or *most* of their relationships are positive. Testers seemed less sure of the value developers placed on tester contributions. Yet, overwhelmingly, developers claim to value those contributions.

Insight: I hope this information helps to shift the perception for testers. Although they might not express it on a regular basis, developers appreciate the value of having testers.

I have not discovered that bad practices are good or that good practices are bad. What I did discover is that our opinions benefit from scrutiny and that a fresh look at conventional wisdom can produce new and powerful insights into the testing process.

Appendix I – Questions from Respect Survey

From the developers' survey:

Do you develop software that is or will be used by external customers?
Possible Responses: Y or N

How many testers do you work with on a regular basis?
Possible Responses: 0; 1; 2; 3 or more

How many testers, with whom you work on a regular basis,
do you have a positive working relationship with?
Possible Responses: All; Most; Some; None

Consider all testers you have worked with at SAS, past or present.
How often have you valued their contribution to the development process?
Possible Responses: Almost Always; Most of the time; Some of the time; Never

From the testers' survey:

Do you test software that is or will be used by external customers?
Possible Responses: Y or N

How many developers do you work with on a regular basis?
Possible Responses: 0; 1-2; 3-4; 5 or more

How many developers, with whom you work on a regular basis,
do you have a positive working relationship with?
Possible Responses: All; Most; Some; None

Consider all developers you have worked with at SAS, past or present.
How often do you think developers value your contribution to the development process?
Possible Responses: Almost Always; Most of the time; Some of the time; Never

Acknowledgements

Special thanks go to Sue McGrath Carroll, Mark Traccarella, Stephanie Tysinger, JoAnna Maguire, Liza Lucas, Audun S. Runde, Ph.D., Cynthia Morris, and all of the anonymous testers who contributed to this paper.

References

Bill Hetzel, 1988. *The Complete Guide to Software Testing (Second Edition).* Wellesley, Massachusetts: QED Information Sciences, Inc..

Roger S. Pressman, 1992. *Software Engineering: a Practitioner's Approach (Third Edition).* New York: McGraw-Hill, Inc.

Elizabeth C. Langston

Elizabeth C. Langston is a Principal Tester at SAS, working in the Testing Process and Research department of the Central Testing Services division. She has more than thirteen years of experience in software testing and test management. She has also served in the United States Air Force as a data automation officer. Ms. Langston has a Bachelor of Science degree in Computer Science from the University of Mississippi and a Master's degree in Computer Science from North Carolina State University.