Food for Thought™

An e-newsletter published by
Software Quality Consulting, Inc.

February 2008
Vol. 5 No. 2

Welcome to **Food for Thought™**, an e-newsletter from **Software Quality Consulting**. I've created free subscriptions for my valued business contacts. If you find this newsletter informative, I encourage you to continue reading. Feel free to pass this newsletter along to colleagues by using this **Forward Email** link. If you've received this newsletter from a colleague and would like to subscribe, please use this **Enter New Subscription** link. If you don't wish to receive this newsletter, use the **SafeUnSubscribe** link at the bottom of this newsletter, and you won't be bothered again.

Your continued feedback on this newsletter is most welcome. Please send your comments to **steve@swqual.com**
e-mail address]

In this issue

In **This Months' Topic** I discuss the lack of common sense in the software industry...

Regular features to look for each month are:

- **Monthly Morsels**
  Hints, tips, techniques, and references related to this month's topic
- **Calendar**
  Conferences, workshops, and meetings of interest to software engineers, QA engineers, and anyone interested in software development

This months' topic

Common Sense Isn't All That Common

Every culture has its own collection of sayings, phrases, colloquialisms, and proverbs. These sayings often provide examples of what you might call common sense. And as I have observed, unfortunately common sense isn't all that common...

One of the most frustrating things about working in the software industry is the lack of common sense. I'm sure many of you have experienced firsthand numerous examples of decisions and actions that seem, well, just plain stupid.

- How many times have feature changes been slipped in at the last minute, knowing that these changes will cause the application to become unstable and regress?
- How many times have decisions been made to release products with dozens of known bugs - knowing that customers will not be happy and that unplanned bug fix releases will be required?
- How many times has the same broken process been used yet again on another project only to achieve the same dismal results?

Here then are some of my favorite **examples of simple common sense** and some suggestions for how you can inject some sanity into your work...

- **"A picture is worth a thousand words."**

  This is one of my favorite sayings because it is so obvious. Humans are much better at understanding graphical information than textual information, especially when we are dealing with complex concepts - such as the features in many software applications...

  Yet, we still try to express this complexity using a language that leads to many misunderstandings. Compounding this problem is the fact that the people writing requirements often have had little, if any, training in using English to express

complex ideas in a manner that is clear, unambiguous, and easy to grasp.

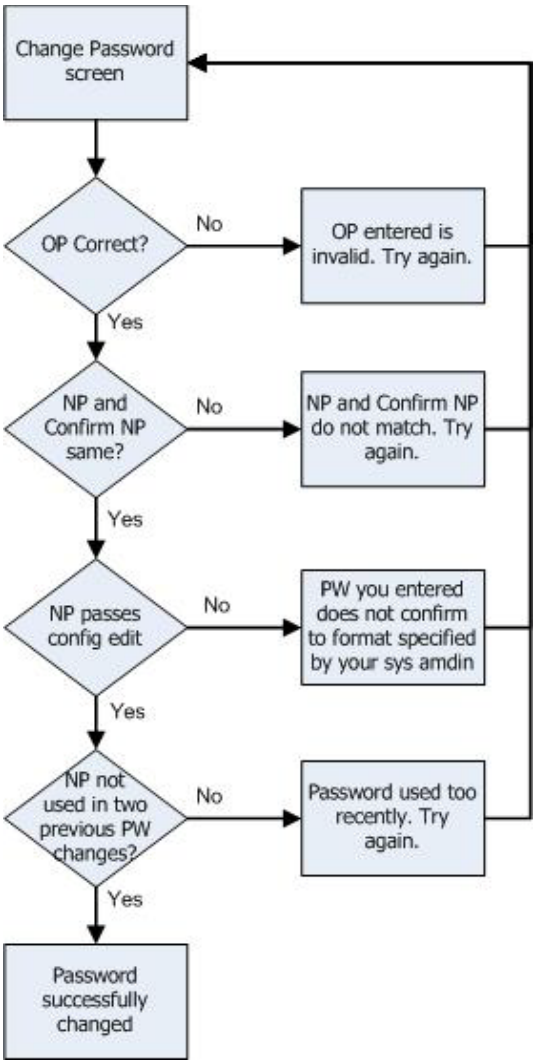
Here's a simple example. The following are the requirements common to many applications that have passwords. These are the rules for changing a password.

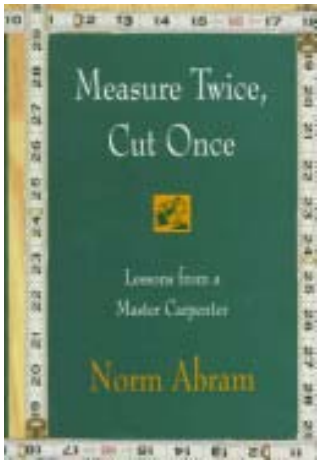
Requirements for changing a password...
(OP = Old Password NP = New Password)

User enters a new password (NP). Application determines if NP meets the following rules.

1. If OP is correct, NP and Confirm NP are same, NP passes configuration edit, and has not been used during prior two password changes, confirm change with a successfully changed password message. Display Message = Password successfully changed.
2. If OP is correct and NP and Confirm NP match but do not conform to configuration settings, a message describing error is displayed. OP, NP and Confirm NP are blank after user selects "OK" on error message. "OK" button brings user to Change Password screen. Error Message = Password you entered does not conform to format specified by your system administrator. Enter a valid password.
3. If OP is valid and NP and Confirm NP do not match, a message describing error is displayed. OP, NP and Confirm NP are blank after user selects "OK" on error message. "OK" button brings user to Change Password screen. Error Message = NP and Confirm NP entries do not match. Try again.
4. If OP is correct and NP and Confirm NP match and pass configuration but has been used prior by user during previous two changes a message is displayed. OP, NP and Confirm NP are blank after user selects "OK" on error message. "Ok" button brings user to Change Password screen. Error Message – Password used too recently. Try again."
5. If NP and Confirm NP match and pass configuration but OP is invalid, a message is displayed. OP, NP and Confirm NP are blank after user selects "OK" on error message. "Ok" button brings user to Change Password screen. Error Message – OP entered is invalid. Try again.

Now here's the same requirements expressed as a **simple flowchart**...





Which is easier to understand? Clearly, the flowchart enables the reader to grasp the complexity more easily and with less confusion than the words. In addition, flowcharts provide added benefits for testers...

- Flowcharts identify all the paths that need to be tested and also ensure that the default or "else" case isn't left out as is often the case when requirements are written in English.

What can you do?

If you find the flowchart easier to understand, then look at your own requirements documents. Do they contain pictures, flowcharts, or other graphical means for conveying information? If they don't, then...

- Take a few examples of requirements from one of your past products. Represent these requirements using an alternative to English -- flowcharts, structured English, or use case diagrams.
- Talk to the folks who write requirements. Show them how much simpler it is to convey complex requirements using these techniques...
- Encourage them to attend a **Requirements Writing workshop**...

- **"Measure twice, cut once."**

  One of my favorite TV shows is the **New Yankee Workshop** where master carpenter Norm Abram builds furniture seemingly without ever making mistakes. I've built some furniture using Norm's plans and made many mistakes. Of course, Norm's mistakes are edited out of the show but he always stresses the importance of getting the measurements right and of paying close attention to details...

  When developing software, the devil truly is in the details - thousands of them. This fact was illustrated in 1999 when NASA's **Mars Climate Orbiter** crashed on the surface of Mars. A NASA review board investigating the crash found that half of the software engineering team thought that calculations were done using the English system while the other half of the team thought calculations were done using the metric system. Ooops! That little mistake cost us taxpayers **$125 million**.

  On the plus side though, since this incident, NASA has mandated the use of Independent Verification and Validation (IV&V) on all projects that involve software. Another set of eyes could have helped prevented this mistake.

  What can you do?

  - If your software uses units of measure, check, double-check and triple-check to make sure they are right and are used consistently.
  - You have to **sweat the small stuff** if you want to ensure your software doesn't crash and burn like the Mars Orbiter. Every detail needs to be addressed, documented, and tested.
  - Attend a **workshop on Software Verification and Validation**

- **"When you throw dirt, you lose ground."**

  I believe this comes from an old Chinese proverb. Makes sense doesn't it?

  When a project fails, rather than trying to learn from the failure, we often look for ways to make ourselves look good, usually at someone else's expense.

  When we play the blame game and point fingers at others, the respect of those throwing dirt is diminished. Eventually, you'll have no dirt left to throw. Maybe then you'll realize that blaming others is not an effective way to learn from mistakes.

  Of all the stupid things software companies do, one that definitely is at the top of my list is making the same mistakes over and over again.

  What can you do?

  - When a project fails, identify root causes in a manner that doesn't lay blame or point fingers.
  - Learn how to perform **Root Cause Analysis** for defects and projects...
  - If you work in an organization that tends to blame people or teams for failures, consider doing a **Project Retrospective** instead of an ineffective post-mortem.

- **"You only have one chance to make a first impression."**

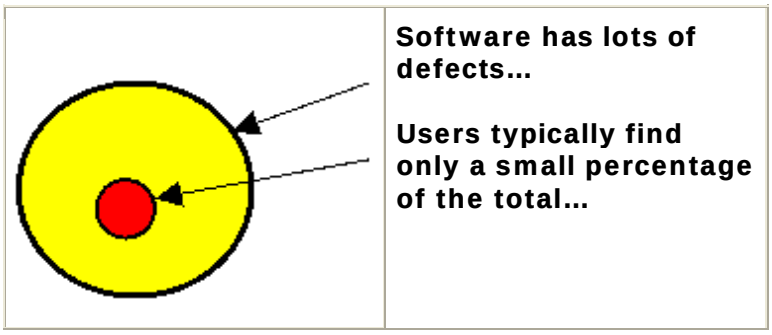
  Whether on a job interview or a blind date, first impressions are critical for the relationship. If you make a bad first impression for whatever reason, it is often very hard to change the other person's impression of you. On the other hand, if

you make a good first impression, you might get a second interview or a second date!

Well, the same is true of software - the first impression your customers have of your application is critical. When customers experience installation problems, they form negative opinions of your software, regardless of how good it may be. The same thing happens with patches, bug fix releases and service packs. Quite often, these releases are not fully tested and as a result, can do more harm than good.

What can you do?

- Understand the installation processes used by your customers and make sure you use the very same processes as part of your internal testing...
- Incorporate **Act Like a Customer™ (ALAC) testing** into your testing process. **ALAC™** tests are based on how **users actually use your product**. These tests are critical in helping to uncover the kinds of defects that users are likely to encounter. Creating **ALAC tests** requires testers with **domain knowledge.**.. a critical testing skill.



- Learn more about **Act Like A Customer™ Testing**

- **"You can pay me now or you can pay me (a lot more) later..."**

There used to be an ad for oil filters where an auto mechanic leans into the window of a car and tells the owner about the importance of changing the oil filter regularly. The owner seemed reluctant so the mechanic says, "Well, you can pay me now or you can pay me later. But if you pay me later, it will be a lot more."

Does this resonate with your last project? Some decisions made on projects are like deciding to not change the oil filter. It may save you a few bucks and a few minutes now but eventually, costs much more down the road...

- **"We never have time to do it right, but always have time to do it over."**

By far, this is my favorite saying because I see this happening so often in our business. Management doesn't understand that **getting it right the first time is the most efficient way to work**. From poorly written requirements, to inappropriate architecture and design, to badly written code, or insufficient testing, we always seem to find excuses to **not** do it right the first time.

As a result, we muddle through projects and deliver something that we know is not going to meet customer expectations. The thinking is that **we'll fix it with a patch release**.

This illustrates what I call the "**patch release mentality.**" In the software

business, there is an assumption that there will always be a patch release. So why not use patch releases as an excuse to deliver lower quality products? It should be obvious that the "patch release mentality" is not cost-effective business strategy in the long run.

With that, I would like to leave you with one last quote...

Quality is never an accident; it is always the result of high intention, sincere effort, intelligent direction and skillful execution; it represents the wise choice of many alternatives.

William A. Foster

'Til next time...

Every month in this space you'll find additional information related to this month's topic.

- **References**

1. Weigers, Karl, Software Requirements, 2nd edition, Microsoft Press, 2003.
2. Kerth, Norm, Project Retrospectives: A Handbook for Team Reviews, Dorset-House, 2001.
3. Rakitin, Steve, Software Verification and Validation for Practitioners and Managers, 2nd edition, Artech House, 2001.

- **Additional Resources:**
  - Karl Wieger's **In Search of Excellent Requirements Course**
  - Steve McConnell's **Requirements Seminars**
  - Norm Kerth's **Project Retrospectives website...**

Every month you'll find news here about local and national events that are of interest to the software community …

- **Software Quality Calendar**

  There are many organizations that sponsor monthly meetings, workshops, and conferences of interest to software professionals. **Find out what's happening...**

- **Workshops Offered by Software Quality Consulting**

  Software Quality Consulting offers workshops in many topics related to software process improvement. **Get more info...**

Software Quality Consulting provides consulting, training, and auditing services tailored to meet the specific needs of clients. We help clients fine-tune their software development processes and improve the quality of their software products. The overall goal is to help clients achieve Predictable Software Development™ – so that organizations can consistently deliver quality software with promised features in the promised timeframe.

To learn more about how we can help your organization, **visit our web site** or **send us an email**.

 Graphic design by **Sarah Cole Design**