QUALITY
Software and Testing

April 2007

The Role of the User in the Software Development Life Cycle

We asked some of the leading people in QA to give us some thoughts about the role of users. Many of them responded.

Thoughts by

Rex Black	James Hobart
Michael Bolton	Cem Kaner
Richard Bornet	Joe Larizza
Rebecca Fielder	Mike Pregmon

TASSQ
Toronto **A**ssociation of **S**ystems and **S**oftware **Q**uality

An organization of professionals dedicated to promoting Quality Assurance in Information Techology
Web Site: www.tassq.org

QUALITY Software and Testing

FEATURES

How to Submit Articles to Quality Software and Testing

Quality Software and Testing provides a platform for members and non-members to share thoughts and ideas with each other.

For more information on how to submit articles, please visit http://www.tassq.org/quarterly.

TASSQ

Toronto **A**ssociation of **S**ystems and **S**oftware **Q**uality

An organization of professionals dedicated to promoting Quality Assurance in Information Technology.

Phone: (905) 822-6645

Email: tassquarterly@tassq.org

Web Site : www.tassq.org

QUALITY Software and Testing

Editorial

The topic of this issue is "*The User*". Our existence as testers and Quality Professionals is dependent on the fact that we create products (software), which someone will *use*. Most of us work with software which has some kind of *user* interface – this is what allows the users to perform their needed tasks. But even those of us who labour in software even if it on background code, such as printer drivers or middleware, somewhere, someone will be able to get some *use* from what we have done. That person is the *user*.

So we thought we would dedicate this issue to that very important person; "*The User*".

When I floated the idea, the first thing that hit me was that this did not resonate with very many people. There was very little energy for this topic. It was as though the user did not register as being important to many of us who are involved in software development.

This struck me as odd, because I believe that the user, in the end analysis, is the only person who really matters in software development. Our whole reason for existence is to enhance the user. The tester's role is to represent the interests of the user, and find errors before the user does. Well I said this to many people, but I did not receive much support for my point of view.

Actually I was directly challenged by Cem Kaner himself. He has contributed a thought provoking piece which takes to task people like me who believe that we testers represent the users. I am sure his remarks will spark much debate.

Rebecca Feidler talks about *Putting the User at the Center of Software Testing Activity.* She introduces us to "qualitative research approaches that can help software engineers, especially requirements analysts and testers, place the user at the center of their efforts." Anyone interested in the user experience and how to learn from it needs to read this article.

We conducted an interview with Jim Hobart, which we have reprinted in its entirety. Jim is a leading User Interface and Usability expert. Among other things, he has been helping NASA re-design parts of their mission control software to assist them in decision making during shuttles launches. Jim gives a great overview of how usability looks and why it is important.

Both Mike Pregmon and Rex Black contributed thoughtful articles on the role of the user.

I put together a tongue in cheek article on why we ignore the user. Michael Bolton in a similar vain asks the question do we really like users, and shouldn't we?

Joe Larizza talks about Vendor Testing where the user is the company who demands that the vendor, whether it is an offshore 3rd party development house or a more commercial vendor, be responsible for the testing.

Finally, I have included a short article I wrote on the various uses of automated testing over and above regression testing.

Our regular features include our humour section with our cartoons. This month we have a series of cartoons poking fun at developers who cannot create useable interfaces.

Also LITruisms (Life and IT Truisms or LITruisms for short), which are some insightful thoughts for you to paste on your office wall. They are sprinkled around the magazine. If you have any to share with us, please send them along.

We hope you enjoy the magazine. Please feel free to drop us a line; we would love to hear from you. And if you have an article where you share your ideas, send it in.

Richard Bornet
Editor-in-Chief

PUBLISHER
Joe Larizza

EDITOR-IN-CHIEF
Richard Bornet

ASK TASSQ EDITOR
Fiona Charles

BOOK REVIEW EDITOR
Michael Bolton

The Role of the User in the Software Development Life Cycle

In the last magazine, we put out a call for articles. Here is the text that we sent out:

"The next topic we would like to address can be summed up in two words: "The User."

"As testers, we owe both our livelihoods and our professional raison d'être *to the User. We build software so that someone will use it. So we all accept that the User is very, very important… or do we? The recent highly successful QAI conference in Toronto did not even have one presentation that focused on the User. A search on the word "user" through the conference program yielded only three hits; not one of these indicated that the User would be talked about in any detail.*

"So we would like to dedicate one issue to the person we are supposedly trying to please with all the work we do.

"We would like to hear your thoughts about the users. Here are some questions we would like to pose to help you think about this topic:

"How important is the user, really?
Why do we not focus more intensely on the user?
Should the user be involved in the Software Development Cycle? How much?
How should we involve the user?
When should we involve the user?
Is User Acceptance Testing important and useful?
How could we make UAT more effective?
Is it the role of the testers to be the representatives of the user, finding the issues before the user does?
Is the tester really the user's advocate?
Should the importance of usability be raised?
Is usability even in the domain of the testers?
Are there any tools out there which could help us with working with users?
Do the users have any recourse for lousy software?

"In answering these questions or any others you would like to pose, please remember the following:

"We really want to emphasize the practical. *What you can* do *is of more interest than some theoretical discussion. We like down-to-earth experiences rooted in real life. People often talk about the importance of the user, but what does that mean in practice?*

"Also we don't mind controversy. If you have a controversial point of view, please send it along!"

We thank Rex Black, Rebecca Fielder, Jim Hobart, Cem Kaner, and Michael Pregman for their contributions.

We then supplemented their thoughts with our own.

Below are the responses we got. We have listed the authors in alphabetic order, just to make things simple.

Again, we are very grateful to the people who took the time and effort to write to us.

Richard Bornet

Considering the Role of the User in Risk-Based Testing

From Rex Black

With almost a quarter-century of software and systems engineering experience, Rex Black is President of RBCS, Inc., a consulting, assessment, outsourcing, and training company, providing industry leadership in software, hardware, and systems testing, for about fifteen years. RBCS has over 100 clients spanning 25 countries on six continents. Learn more about Rex and RBCS at www.rexblackconsulting.com.

If you read my last TASSQ article, you know I'm a big fan of risk-based testing. In this article, I'll consider the role of the user in risk-based testing.

Remember that risk-based testing is about tailoring your testing to allocate effort and sequence tests for potential problems based on considerations of impact and likelihood. Risk analysis and assessment helps you identify these potential problem areas and assess their levels of risk. However, the analysis and assessment is only as good as the people involved.

When identifying quality risks, we need to consider what quality means. Who knows this better than the user? When assessing the level of risk associated with a particular potential problem, we need to consider not just the technical aspects (which influence the likelihood of a bug), but also the business and operational aspects (which influence the impact of a bug should it exist). Who knows the business and operational aspects of a system better than those who will use it?

Unfortunately, project teams who employ risk-based testing do not always involve users; which leads to untested risk areas, over-testing some areas and under-testing of others, and improper sequencing of tests. Why don't project teams always involve users in risk analysis and assessment? I have seen three main reasons.

First, some project teams don't approach risk-based testing properly. They see risk-based testing solely through the lens of technical considerations. When testing is seen as mostly concerned with finding as many bugs as possible, without considering the confidence building and risk-mitigation roles of testing, then risk-based testing becomes monomaniacal.

This problem is easy to fix: Change the project team's approach to risk-based testing to consider both likelihood and impact. Training can help with this, as can reading a book like my own Managing the Testing Process or Rick Craig's Systematic Software Testing.

Second, some project teams have limited access to users. This happens when building software for mass-market sale. It also happens when users are geographically separated from the project team.

This problem is somewhat harder to fix: The project team must identify user surrogates like business analysts, sales, and marketing teams. These people must act on the user's behalf in risk analysis, and must remember to validate their assumptions with real users as often as possible.

Third, some project teams have organizational challenges to involving users in risk identification and analysis. IT stakeholders sometimes worry that the user community will be alarmed by a frank discussion of risks, leading to loss of support for the project or re-opening of debates about the project's viability.

This problem is harder to fix: The users who will participate in the risk analysis must be helped to understand that the very act of analyzing and assessing risks will allow for the mitigation and management of those risks. Risk-based testing makes projects less risky, not more. Convincing and educating users requires a strong champion for risk-based testing, who can be a technical or managerial leader in the organization or an outside consultant.

The role of the user in risk-based testing is at once essential and challenging. Don't let the challenges frustrate you, though. The challenges can be overcome, and must be overcome, to achieve success in risk-based testing. The alternative is an approach too risky to consider!

Users We Don't Like

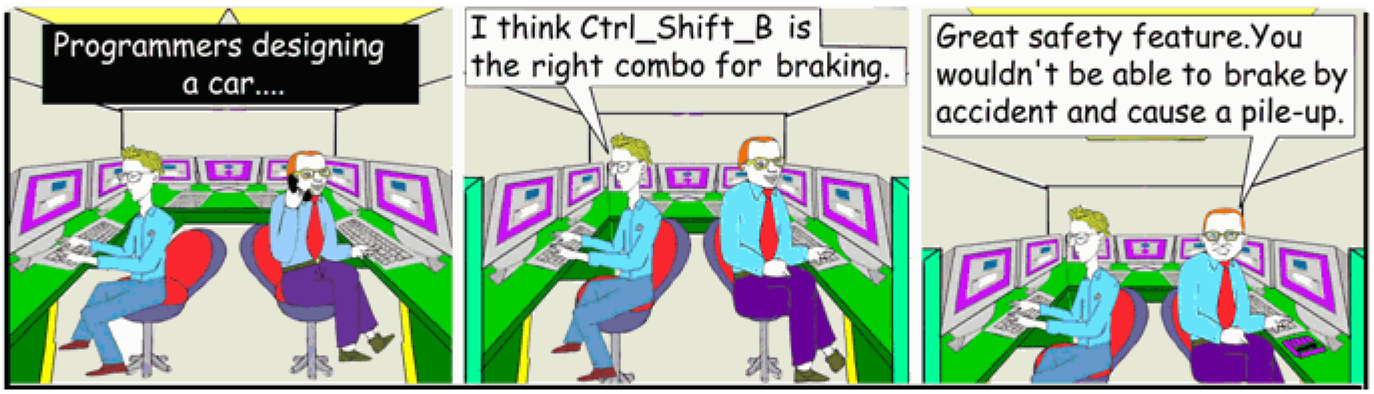
From Michael Bolton

Michael Bolton is the co-author (with James Bach) of Rapid Software Testing, which he teaches in Canada, the United States, and all over the world. Michael is also the Program Chair of TASSQ. You can reach him through his Web site, http://www.developsense.com. /

When we think about the user, I've found that it can be helpful to start thinking about users that we don't like--disfavoured users, as Cem Kaner calls them. We definitely don't like hackers, so we want to frustrate them. If we're developing an application in which accountability is important, we probably don't like embezzlers, so we need to think about audit trails and so forth.

When I presented this notion in a Rapid Software Testing class a couple of years back, I was trying to get people to think about these kinds of disfavoured users, but to my surprise the conversation went a different way.
"What kinds of users don't we like?" I asked. To my surprise, "People who don't read the manual," was the first answer. There were several laughs in the room. I remembered Marshall McLuhan's remark that every joke contained some kind of legitimate complaint, so I let the list continue, with more laughs. "Novices!" "People with old computers!" "People with NEW computers!" "Developers!" "People who DO read the manual!" "Marketing people!" "Senior managers!" "Other testers!" Pretty soon, we had catalogued every member of the project community. In fact, our list of people that we didn't like was really a list of people that we /should/ like and who should like us--but if they found that the product disappointed them in some way, that would spoil the relationship. So, starting by considering the world in a topsy-turvy way, we had quickly come up with a good list of people whose needs and interests we ought to consider.

This led to another idea, which was more sobering: most of us claim to like disabled people, people from other countries and cultures, people who need help from our products. We claim to like them, as well we should. But do our actions in the world--especially as designers, developers, and testers of software--reflect that consistently?


Programmers designing a car....
I think Ctrl_Shift_B is the right combo for braking.
Great safety feature.You wouldn't be able to brake by accident and cause a pile-up.

The User

From Richard Bornet

Richard Bornet has been in the software business for over 20 years. The last ten he has spent running various testing departments and creating innovative approaches to improve testing. He specializes in test automation and is the inventor of Scenario Tester and co-inventor of Ambiguity Checker software. He can be reached at rbornet@eol.ca. or by phone at 416-986-7175

A major part of my testing career has been spent on testing in-house systems. In my experience, the user generally seems to be an irrelevant cog in the machine. Users are almost never included in any decision making; usability issues are not considered important enough to have any formal time or resources allocated to them; very little time, if any, is spent on any formal usability testing.

This lack of involvement or caring for the user has always astounded me. Would it not be logical to assume that if the developed software is not useable and / or does not meet the users' needs, then there is a serious problem with the software and that the problem will translate into increased costs? Increased costs could come from rising support and training costs at one end, to the software not being used, to mistakes being made by the users, and to the software being junked at the other end. Would it not make sense to make sure, as much as possible, that the software is useable and does meet the users' needs?

The answer is, "Of course". So why do we not do it?

Before we get into that, that let me describe how it could be done. The technique which could solve it is fairly straightforward. It works something like this.

- Gather enough requirements to allow you to build a prototype. These requirements need to focus on the positive scenarios, and they should mimic the actual activities that the business will perform.

- Build a series of working prototypes or models to represent how the business will perform the tasks. There are tools that allow for very quick prototyping. Build several competing prototypes so more then one model is tried out.

- Bring in actual users to try out performing the business scenarios. Watch how the users perform their tasks, and get feedback.

- Adjust the prototypes or create new ones, until the proper interface and functionality are finalized.

- Create the rest of the screens for the application.

- Test them.

At the end of this process, a working model of the software is completed, and the developers can then create the actual application, without having to worry about the interface.

Sounds simple? It is. Getting the right interface, though, takes some skill.

So why do we not do it? I have not only pondered this question, but in many organizations have gone to battle for getting the interface right. And, in most cases, I have not gotten very far. It was like I hit a brick wall. So after much pondering, I have come to the conclusion that the reason for this is that we are human; and although we profess that we are logical, upstanding individuals who have the best interests of the organization in mind at all times, the reality is that we play out interpersonal human scenarios, not only in our personal lives, but also at work. Behind this is the need to not admit to ourselves our own weaknesses, and our need for control.

Here are the different scenarios that play out:

1. **Don't admit that you are lost, and definitely don't ask for directions.**

This is the classic stereotype of an alpha male. They deny that they are lost; insist that they know where they are, and absolutely refuse to ask for directions. They consider this a sign of weakness, failure and humiliation. Well, this is the type of scenario that plays out too often with developers, managers and business analysts when it comes to designing interfaces. Very few developers I have met are really good at designing wonderful interfaces. The rest bluff.

2. He who controls the interface controls the project.

This can be illustrated by an anecdote. In the early days of the web, a developer designed a customer entry screen. This was written for external customers, to allow them to enter data into this company's database remotely. One of the fields was birth date. The problem was that it only accepted one date format. A user would enter the date, and when the page was sent, an error message would come back, stating the date was in the wrong format and showing the correct format. The bigger problem was that the page was slow, and it took quite some time for the page to return with the error message.

The user representative was in the meetings and was quite frustrated with the delay on what was fundamentally a data entry system. I suggested that we just add the format below the edit box, so a user entering their birth date would know how to enter it; a purely cosmetic change. The user liked the solution. You would think, "Problem solved".

Well, that was only the beginning. The programmer refused. He actually said to the test team, "I will make no changes to the application (read HTML page), only changes to the database". When he was pressed to make the changes, he said, "Will you take responsibility at this late stage to changes to the application?" When the test team said that they would test thoroughly, so he could relax, he still refused. The issue was escalated to the product manager, who took several days to ponder this. He finally sided with the developer. It was the after-affect of this that was the most interesting. The user went back to her company completely disgusted; the test team had a large drop in morale and basically gave up on this project. And I have a great story to tell on how pathetic it can all be.

This was not a battle over yyyy/mm/dd but a battle over control, or at least that is how the developer perceived it.

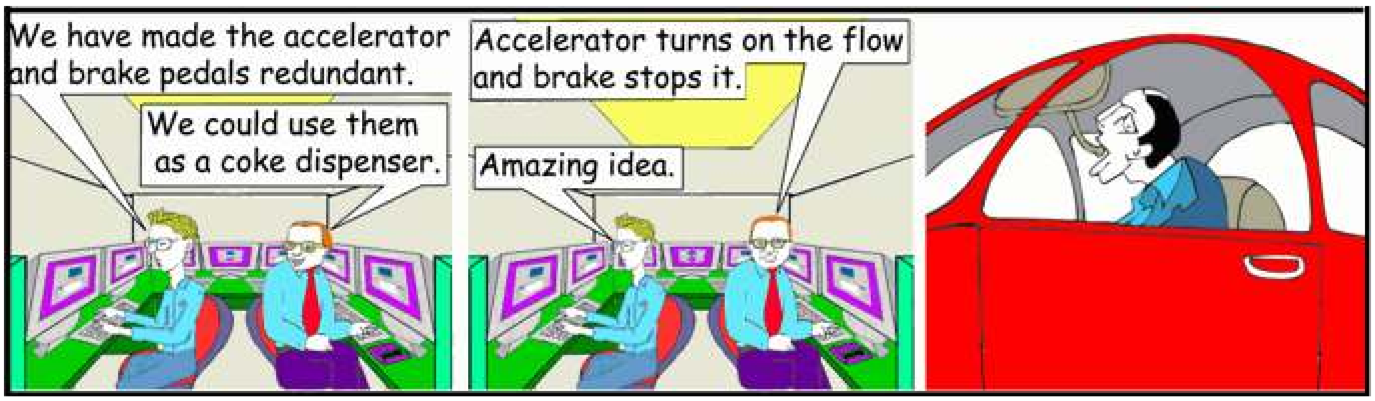
It is true that he who controls the interface really decides how the product will function. If we let a designer and users create the user interface, then the developers are only the engineers who make it happen. Houses are designed by architects; the builders just make that vision a reality. They do not change the vision, they do not decide on the vision, the architect does that. Software is the only major human activity where we allow the designer and engineer to be the same person. In nearly all other products, the designers are separate from the builders. This will eventually happen in software development, and is happening, in web design where real designers are often used. It is a stage of evolution our industry is going through, although not fast enough for me.

3. Everything must conform to the plan no matter what the consequences.

I cannot believe how many times I have phoned customer support for various software packages to complain about how a piece of software works. Almost never do I get a person who says "That's a really good idea and I will champion it for you." Even if the person says "That's a really good idea." I even sometimes have to insist that the support person enter my problem in the database.

For example: My cell phone does not work in my kitchen and I live downtown. I phoned the provider and reported this. I could not believe the response from the customer service agent which could be summed up as "Tough, we have put our network in and we won't be doing anything about it." In other words, "It is not in our plans.

The company has made decisions on what is to happen with their network and that's it. I remember one commercial defect tracking system had a really annoying user interface issue. The change was pretty straightforward. I talked to a senior manager who developed the system. It did not matter how good an idea it was, it was not in their plan.


We have made the accelerator and brake pedals redundant.
We could use them as a coke dispenser.
Accelerator turns on the flow and brake stops it.
Amazing idea.

Well we all know what happens to planned economies.

4. You are just children and I, the adult, know better.

This may be the way some people bring up their children, but is this a way to treat the user? In many cases, this is exactly the way that users are treated. My wife has an expression "If you don't have what you want, then you better learn to want what you have." She says this to our children. What's the difference between that and "Here is how the software has been designed, now learn to use it as we have designed it"? This is fine, we do have organizations, and people should know their place, but who is more important to the corporation - the user who represents the company to the public, or the developer? The manager who makes decisions for the direction of the organization, or the IT product manager? Especially in an organization where the basic business is not software. Maybe IT has forgotten what is the purpose of the organization?

Of course, the best arrangement would be one of cooperation and enhancing and empowering all the parties involved. But too many individuals did not grow up in structures which taught cooperation and creating win-win situations. So now they play out this inability to work together in their adult lives.

5. Everyone is an idiot except me.

I am amazed at how many people have this attitude. I even catch myself exhibiting this sometimes, so I guess we all do it. The problem is that it has some serious repercussions. If you know so much more, then why listen to another? As Q summed it up in Star Trek, "It is hard to be a good team player when you are omnipotent." One may smile, but I have heard many developers say that they get no useful information from the user, so it is waste of time to talk to them. It never occurs to them that the problem may be that they do not know how to ask the questions to get really useful answers.

6. Time is precious and we don't have enough time.

The best way to illustrate this is to give another anecdote. We were testing a piece of software to enter products. The application arrived on the testers' desktops, and this section involved two screens. The screens just did not want to work. After two weeks, it became obvious that the fundamental design of the screens was wrong. The developers based their design on one thing they had heard from the business, and in order to implement this, they designed a series of pretzels. The problem could have been solved by giving the users the right to sort on a table by clicking on the headings, and not restrict how the products were entered.

I complained and was told that it was too late in the development life cycle to make any changes. The screens continued not to work. One morning, in a boring meeting, I re-designed the two screens in an hour. I showed the design to my boss, and he smiled and agreed that this would be a major improvement and would solve all the problems they were having. Would he take it further? Not if he wanted to keep his job. He advised me to do the same. I did not take his advice, and did try to escalate this. I was informed that it was too late in the development life cycle, and that time was of the essence, so changes could not be made. So I asked a good developer how long it would take to create these screens and test them. I was told that even with some unforeseen issues, at most two weeks. To cut a long story short, four months later, they finally got the original screens working so they actually worked. After all this work, the screens still did not really meet the needs of the business.

Sound familiar?

7. What we do is so important that we can't find the time to waste talking to you.

Before I totally blame the developers, let's look at the business. Just as often, I have seen the developers or designers come to the business asking for their time, and being told that the users are so busy, and what they are doing is so important, that they really do not have time to waste talking to the lowly designers in IT. What is even more astounding is that the software is being designed to empower and enhance the very important activities that those individuals are performing. Excuse my language, but this is just self-serving arrogance.

My reaction is, if you don't have time, then the problem is obviously not serious enough to waste the time of the

LITruism

Too often, IT people cannot get the time or attention of users because what the users are doing is very important and their time is valuable. This happens even though the IT people are building application(s) to empower the user. So if the software is important, then those important individuals need to find some important time out of their important schedules to do something that is important. Like get the software right, so they can continue to be important.

developers. Figure out something important that they can be working on. If this software is important, then those important individuals need to find some important time out of their important schedules to do something that is important. Like get the software right, so they can continue to be important.

8. The love affair with mediocrity has no bounds.

I have often been told that "they appreciate my desire for perfection but time is limited". I agree that perfection is a time consuming activity, but mediocrity seems to take twice as long.

This is a major problem. The question, "Is this good enough, or can we make it even better?" does not get asked enough, if at all. This is just a fact of life. We put up with poor service, poor relationships, ineffective processes and structures, so why not poor software?

Wanting to create a product which rises above the level of mediocrity is a question of attitude. If successful it's gives people both a sense of fulfillment and a sense of honour. And you get a much better product usually completed in less time.

9. My reality is the real reality because that's where I feel comfortable.

This seems to be a universal affliction. We all have a world we live in which we know and feel comfortable with. We gravitate to activities and people where we can behave in a way that does not threaten our egos. We also want to live in this world and not live in someone else's world. So if we enjoy and like talking about football or cooking, we will find friends with whom we can share our interests.

It is not that much different at work. Programmers like to talk with programmers. Testers hang out with testers. But it is more than that; programmers think and want to write code, business analysts think and want to write requirements documents.

It is the next stage which is so interesting and that is that we really don't make the effort to truly understand another's world. It's too much effort to leave our little cocoon. So the developer really is not that interested in what the user's world is. The IT manager needs to manage and control, it's not part of his or her world to spend a lot of time thinking about what it means be able to service a customer properly.

Most of the people who create software and user interfaces are not users of the system. So they either cannot relate to the user experience; do not want to admit this to themselves; and if they did may not want to do anything about it. It is really not part of their lives.

I go to neighbourhood meetings where people argue for hours about the possibility that a few extra cars may travel through the neigbourhood and how to prevent this hypothetical blight. But when I suggest we should have a meeting about the future direction of our city, there are no takers. I am sure there is even less interest in talking about Darfur, North Korea or the mass deaths in Africa caused by malaria, poor drinking water and AIDS and what our neighourhood could do about it. It is not because these are bad people; actually the exact opposite is true; these are very dedicated, concerned, interested, good people. They talk about the cars because they can relate to a few extra cars on a street in front of their house.

10. Most of us don't listen that well but we would never admit this to ourselves.

I did a survey and asked people how good a listener they feel they are on a scale of 1 to 10. The average score was an eight. But most of these people did not feel that others listened so well. On average they rated other people a four. But each one of us is "an other" to someone. I am not sure we listen so well. We often don't even hear what others are saying and if we hear the words we will interpret it to match our way of thinking about it, which may or may not be what the other really said.

Many years ago, I was usability testing a piece of software I had designed. We had a menu called Format which at the time was a new concept. Under the Format menu we had an item where a user could change the look of something.

In the usability study people were asked to make that change. They would look under all sorts of menus but not Format. They would search through and verbalize "I need to change…". I was very upset and spent a great deal of energy trying to figure out how to get people to choose the Format menu. If I had only listened! The users told me what I needed to do. Just rename the Format menu to the Change menu. A 20 second change. Done. But I was not listening.

So even if we could get users and developers together, would they actually listen to each other?

So what can we do about improving the user experience and making the software interfaces more productive and efficient?

Here are some quick suggestions:

A. User interfaces matter; usability matters.

This is not a secondary issue. To borrow from Vince Lombardi, "The User Interface isn't everything, it's the only thing." This is what you get right, so you bring the resources needed to get it right. Then you test the hell out of it. And then you get it more right. The time it takes to get it right will pay major dividends, both in productivity of the end users, and also in shorter development time.

B. You need a champion.

Jim Hobart told me a story about Steven Jobs when he came up with the idea of the iPod. Engineers would bring prototypes to him, and he would go into a rage. He would throw the prototypes out of the door with the engineers after them, and yell, "Now design what I want."

Now, if you are CEO of Apple and have an impressive track record, then you have clout. But organizations need to have individuals who can say, "This is not good enough and this is what you need to do to improve it". And not have those individuals be over-ruled by managers who got there because of their talent in playing politics.

C. So give the power to the architect

In most mature industries, there is someone who has the creative vision and designs the product. That person or team also directs the building of the product, but does not do the building directly – someone else does that. For example, when we build buildings, we have the architect vs. the construction company. In the movies, we have the director vs. the actors. In the automotive industry, we have the designers vs. the assembly line workers.

When the product is relatively small, the "creator" and the "constructor" tend to be the same person. For example, if I were building a small tree house for the kids, I would probably design it and build it myself. But these roles should separate as the industry matures and the products become bigger.

In the software industry, we still sometimes have the "we're building a small tree house" attitude. The person who builds the product also designs the product. We need to recognize that our software projects are bigger than that. The software architect needs to drive the creation and retain the creative control, while the developers write the software.

The "creator" has a mandate to produce something someone else wants to use, buy, or see. Their success is ultimately measured by how their creation is received. The software architect's success, then, is measured based on increased user productivity, on whether the users can perform new tasks, and on user satisfaction.

D. Develop a service culture

Software development is a service that IT provides to the users. Not the other way around. The users are not irrelevant cogs, they are the customers. I have been on projects where the developers were shown video recordings of users struggling horribly with the masterpieces the developers had created. Generally, the developers got extremely upset. A few developers blamed the users, but most realized that they hadn't done a very good job, and wanted to improve the software. Most developers want someone to use their creation so as to become more effective.

But it is more than that. The attitude that IT is really providing a service needs to permeate an organization. They have to constantly ask, "How can we be of service?" They really have to be dedicated and motivated to implementing something that will be of significant benefit to the user. It needs to be their raison d'être.

OK, accelerator pedal turns on Coke, brake stops it.
So what would we use the clutch for ?
We could use it as a toggle for Diet Coke.
Great idea!

E. Force the developers and designers to use their own creation.

If possible, get the people who create the software to actually have to use the software. They could start by using the old software for long enough that they experience the issues. So if it is data entry software they are creating, let them enter data for a week.

If they can build a good enough prototype, let them then mimic the business for a week.

If they do this long enough, they become users.

F. Figure out how to give the user a vote

Software, which is commercial, has a measure of success. Someone buys it or does not buy it. If there is competition, then there is some incentive to make sure the software has excellence, and meets the needs of the user. For web sites, the measure is also fairly quick. People go to your web site and buy stuff, or they do not. They use your web site, or they phone support. This can all be measured. The users vote with their mouse.

Figure out how to measure these activities and give some incentive for goals to be reached. For example, have a bonus structure, but subtract from that amount the money spent on customer support and training. The more support is needed, the smaller the bonus.

Create measures of user satisfaction and then reward the designers and developers based on the results of the measures.

Create a web site where users can bitch and give suggestions about the product. Make sure that this is treated with the respect that it deserves. And make sure that the designers and developers see it.

G. Put the Acceptance in User Acceptance Testing

Too often I have seen that User Acceptance Testing does not amount to much. Users are expected to come and bang at the keyboards for a few hours and then… nothing. One development manager told me, “We do it as a courtesy.”

I have worked on other projects where Testing and User Acceptance Testing actually had sign-off rights. In this project, the users who worked with the testers pushed back a lot. They would ask for changes to the GUI. The developers would bristle. The Director of IT got everyone into a room and informed the developers that they had to implement the changes the Users wanted and if the developers were late they would be held personally responsible and there would be consequences. I’m actually giving you the polite version - he was much blunter in his vocabulary.

In the end, it actually saved time. Developers ended up working with the users while they were designing the screens, so when they got around to coding them, they did not have to waste time re-designing them. It all turned into a win-win situation.

But this only happened because the user / tester got some clout.

Conclusion

This article was written in a somewhat provocative, “let’s get a reaction out of the reader” tone. This was on purpose. Reaction equals interest and debate, and that’s the goal.

So let me take a break from knocking various people and ask the question, “Have I ever seen projects where the user and the user interface really mattered, and the software team really created something quite extraordinary for the user?” The answer is, Yes I have. Not the majority, but enough.

What differentiated them from some of the others? After scratching my head to try to find some commonality, I finally came up with two events, where at least one was present.

The first was that the development team actually used the product. They actually looked at the software like users, and thought about what would make their personal experience with the software better.

The second was that the software development team cared. I don't mean just about the project and the software. They cared on a very personal level about each other. They cared about the quality of what they were producing. They cared about delivering quickly and not escalating costs. They cared about raising the bar. They cared about producing something wonderful. They cared about the user and the user experience. They cared about what people had to say. They cared that people did their best. They cared that people felt good and had a sense of belonging.

This attitude of caring eliminated most politics, so many of the issues that I mentioned above did not happen. These were wonderful projects to work on. I don't know exactly how you build this, except by being caring yourself. But I do know that politics can wipe it out and effectively suppress it.

These are my observations and my two cents’ worth. I hope you find it helpful.

Putting the User at the Center of Software Testing Activity, Part One

From Rebecca L. Fiedler

Rebecca Fiedler recently completed her Ph.D. in Instructional Technology. Her dissertation research used a social science model and social science methods to examine the impact of a new breed of software tool on the user operating in a high-stakes context. Her work is a demonstration of how these methods can be used in the software engineering field in areas of requirements analysis, test design, and the evaluation of software tools adoption. Although Becky's main professional interests lie in online education and using technology in education, she also hangs out with members of the Context-Driven School of Testing at places like the Workshop on Teaching Software Testing (2005, 2006, 2007), the Workshop on Heuristic and Exploratory Testing 3 (2006), and the Workshop on Open Certification (2006). Becky can be reached at becky@msfiedler.com.

This report introduces readers to qualitative research approaches that can help software engineers, especially requirements analysts and testers, place the user at the center of their efforts. Part 2 (to be co-authored with Cem Kaner) will focus on qualitative methods and tools that can help organize and analyze a large, heterogeneous mass of data of the kind you might collect from project documents, product reviews, and user observations (usability tests, beta tests, tech support logs, etc.). Today’s article (Part 1) introduces a theoretical framework for interpreting this type of data and designing studies to gather it. The report extends an observation of Cem Kaner’s (2004; 2006) that much of what is interesting in software testing looks more like social science than like engineering.

The paper begins with a case study of real people using actual software to complete a high-stakes task. The case study serves as the focal point for the rest of the article as readers become acquainted with the model and think about ways to apply this model to testing and engineering efforts.

Context of the Case Study

The convergence of widespread Internet access and web-enabled databases has contributed to the emergence of a new breed of web-based tool called an electronic portfolio. Many college students who use these systems liken them to an "artist portfolio" and focus on collecting the best examples of their work to demonstrate their accomplishments during interviews. In contrast, program administrators typically see these portfolio tools more like an investment portfolio that tracks performance over time. To administrators, electronic portfolios provide a means to collect and aggregate data about programs and institutions to cope with the accountability demands of today's society.

Accrediting organizations examine higher education programs in fields as diverse as art, counselling, health, education, engineering, science, computer science, and hospitality management. Some accrediting bodies endorse entire institutions. In this context, accrediting activities are mission-critical and many institutions use data collected through electronic portfolio systems to gain and retain this important accreditation.

Electronic portfolio tool developers have been successful at marketing their service to satisfy accreditation needs at the department and college level. In their marketing efforts, they stress the ease and convenience of data collection and reporting for accreditation purposes. Once decision makers decide to adopt and implement a commercial electronic portfolio solution, they require each student to purchase the subscription (approximately $100) from the local bookstore or company website. There is no direct cost to the adopting college or department, but successful implementations may incur costs for additional equipment or personnel. For students, the initial subscription lasts for three years. After the initial subscription lapses, renewal is $50 - 60 per year and is paid directly to the company. This business model relieves cash-strapped higher education institutions of the financial burden of this component of their accreditation initiative making this class of tools an economical and attractive option for administrators. The funding model also provides a steady stream of income to electronic portfolio tool developers as new students enter programs every year.

This confluence of events - changes in the technological landscape; demands of accrediting bodies; a business model that shifts costs to students and offers a steady stream of customers to tool developers - has attracted many electronic portfolio tool developers to the educational software market leading Cohn and Hibbits (2004) to call electronic portfolios "higher education's new 'got to have it' tool" (p. 1). Surprisingly, teacher education programs are often on the leading edge of portfolio adoption for program accreditation.

The Case of VendorBuilt College

This case study is drawn from a teacher education program at VendorBuilt College (VBC)[1], a small, liberal arts college in the Southeast. Students, faculty, and program administrators at VBC are typical users of this tool as they work in the high-stakes environment of program accreditation. Their challenges are similar to those any other user group faces as it works with this class of software tool.

All students majoring in Education must subscribe to the iNetfolio web-based electronic portfolio service. They use iNetfolio to compile work products assigned from courses throughout the program into an electronic portfolio for faculty review at key points throughout their program. Along with each work product, students are required to include the specific feedback received from their professors and their final grade on the assignment. They are also required to use a reflective writing entry to document how these work products demonstrate their achievement of specified state and national program standards. Finally, they must include scanned images of a variety of forms and letters documenting completion of required activities such as classroom observations, state exams, and their college or university transcripts.

At VBC, faculty introduce the portfolio tool, the extensive portfolio requirements, and a faculty-designed portfolio template to students who are just beginning their programs of study. Responsibility for completing the electronic portfolio rests with the students who are expected to work independently over the next two to three years to prepare their portfolios for a review one semester prior to graduation. At this time, they enrol in a class designed to prepare them for their student teaching internship but much of the instructional time for this class also focuses on finalizing (or starting for the procrastinators!) their portfolios so they can proceed to student teaching and subsequent graduation.

Students shared the following experiences and anecdotes[2] in a series of interviews and thinkaloud work sessions:

CATHY: I have Dr. Lancaster for two classes and she assigns these big, huge projects and both of these projects are required to be included in my portfolio and she knows it. But she won't let us turn our projects in electronically. She makes us print them out and everybody knows she likes all of this creative stuff and we have to do all this "cutesy" stuff to get a good grade. Then, I have to do just as much work all over again just to reformat everything to go into the portfolio. I don't have time for this. Most of the time the scanners in the lab aren't working and so it's really hard to scan the things that need to be scanned. I did a lot of the project using Microsoft Word, but when I try to copy it from Word and paste it into iNetfolio, everything just goes crazy. I end up with all of these spaces that I didn't put there and sometimes it takes away some of my formatting and it adds other formatting and I spend hours just trying to clean that up and it just doesn't work. This was a 36-page paper and who has time to retype everything? I don't.

MALLORY: The professors here make us include these certain projects and we're not allowed to switch them out for others that we like better. I want to use this portfolio when I go on an interview and some of the things they want me to include are bad choices for me. They keep telling me it's a good idea to use this portfolio on interviews and then they require me to include these projects. And I just don't want to show this to a potential employer because it wasn't very good. There was a lot going on in my life when I did that project and I did a terrible job. I'm embarrassed to show it to anyone. I can do much better and that's what I want to put in my portfolio and I think that should be allowed. And do you know what else? There's no way to burn my portfolio to a CD so if I want to take it on a job interview I might not even be able to use it if there's no Internet connection in the interview room. What happens if iNetfolio decides to go down during my interview like it did this weekend? Then I will look like a fool. I also can't make a backup of my files or keep a copy after my subscription runs out. That's just not right. I should be able to get my stuff.

ANNE: The future use, and probably the biggest use, for this portfolio is going to be as a template for other people coming up and doing this portfolio class, because I already know I've shared it twice with students within the past few days saying they're going to need it later and to use mine as an example. So that's the biggest use it's going to get. I know it really helped me to have access to two portfolios from older students and I hope mine will help the younger students.

HANNAH: Look! Look! I have two pictures in one section and I didn't think iNetfolio could do that. How did I do that? I don't know how I did that! Everybody, come see this!

[1] VendorBuilt College, iNetFolio, and all names in this article are pseudonyms.
[2] Student quotes are based on data collected as part of my dissertation. The quotes have been synthesized and, although they are not from any specific person, they do reflect actual concerns of real students using an electronic portfolio tool.

ANNE: I was just so stressed out about all of this. You see, I have to have this done because it's a graduation requirement and I felt like I was behind and had to get caught up. The service was running really slow and I didn't get everything done I needed to get done before the computer labs closed the other night. So, since I don't have Internet access at home, I went home to get my roommate's laptop. She has a wireless card and the library has wireless access. So I came back in the middle of the night and I went and sat on the library steps after it closed so I could pick up the signal and work on my portfolio. It was really creepy because this scary-looking guy came by. I kind of moved so he wouldn't see me and, lucky for me, he didn't. After he was gone, I stayed a lot closer to my hiding place, but I kept working. I had to. I got everything done, too.

ALICIA: I've learned a lot about HTML from using MySpace. I don't know if you're familiar with it or not, but I think it's probably taught a lot of our generation how to do HTML. I can put on backgrounds and pictures and make links in my profile. I spend a lot of time using MySpace and I don't understand why iNetfolio won't let me do those same basic kinds of things.

ANNE: Today I'm going to try something I learned how to do in MySpace. I absolutely detest the way iNetfolio shows pictures. I want to put more than one picture in a section and iNetfolio won't let me. So, I'm going to try to edit the HTML so I can use <img src> tags and FreePhotoHost [free image hosting service] to see if I can make this do what I want it to do.

TUCKER: My professor graded my project and filled out the rubric or grading sheet using the iNetfolio tool. I'm required to show that grading guide in my portfolio and I don't know how. The professor said she doesn't know how either except for me to print it out, scan it, and then add it as an image. That doesn't make sense to me. I want to see if I can figure out a better way because I have to do this – like 30 times – and I don't have a scanner at home. Between my classes and my part-time job and trying to have a life, I don't have time to come to the computer lab hoping that (a) the scanners are working and (b) that I can actually use one when I need it.

JUSTIN: I have iNetfolio's tech support number programmed on my speed dial.

RAFAEL: You need to understand the time that's involved in it.

In his interview, Rafael went on to point out that problems crop up and little things add up to a lot of time: time to do the work product, go match it to a state standard and write it in the words of the standard, write the reflection, do the revisions. There are 12 work products and all of the revising and scanning (and image resizing) that go along with each of them. At our interview, he was only days away from graduation and had spent seven hours the previous day trying to scan documents and had been on campus for another three on this day. His chief complaint at this point is that there's a disproportionate amount of stuff to be scanned: many pages of evaluations, 12 practica, six observations, 12 grading guides, and a picture on the splash page.

The Cultural Historical Activity Theory Model

Modeling is one of the core activities of software testing. One group of testers calls their approach "model-based testing" (http://www.geocities.com/model_based_testing/) but state models are just one class of test-useful models (see http://en.wikipedia.org/wiki/Model-based_testing). Read the writings of James Bach (http://www.satisfice.com/tools/satisfice-tsm-4p.pdf), Michael Bolton (http://www.developsense.com/articles/Elemental%20Mode

How would people be able to mix regular Coke with Diet Coke?
They will have to learn to press their foot down at a right angle so they can press two pedals at the same time.
Great, we can offer a training program

ls.pdf), Elisabeth Hendrickson (http://www.qualitytree.com/feature/apwatw.pdf), Jonathan Kohl (e.g http://www.kohl.ca/blog/archives/000176.html), and Mike Kelly (e.g. http://www.testingreflections.com/node/view/2823) and you'll see a steady stream of discussions of models that help testers make/interpret observations quickly and generate new ideas for testing.

Activity theory has been around in one form or another from the early 1900s. Proponents of activity theory recognize that "doing something" involves both context and purpose. In terms of user-centric software engineering and testing, this means users of software are trying to accomplish specific tasks while embedded in an environment or working context that may help or hinder them from completing their tasks. Understanding the user context helps software developers build better software and design more meaningful tests.

This section lays out a basic description of a specific activity theory model developed by Yrjö Engeström (1987). Called Cultural Historical Activity Theory (CHAT), the model reflects the role of a community and the history of that community in shaping activity and how activities are performed. We can use this model to frame a discussion of the electronic portfolio activity at VendorBuilt College and generate testing and development ideas for iNetfolio bug fixes and new releases. Specific examples from the case study illustrate the key ideas associated with the model.

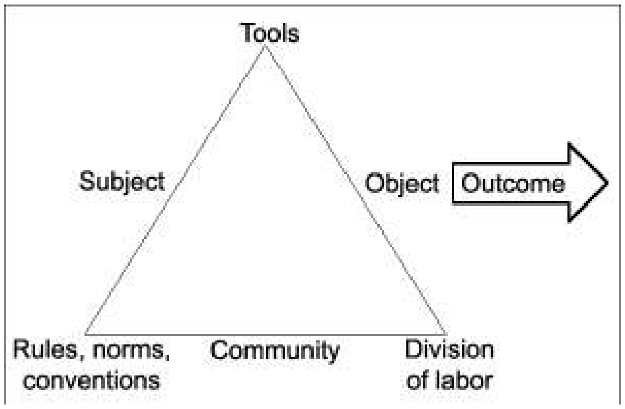


Figure 1: Model of Cultural Historical Activity Theory proposed by Yrjö Engeström (1987)

The CHAT model is represented by a triangle. The first task for using the model is to determine the "unit of analysis." If the unit is too small, it will be meaningless; too large, and the analysis is impossibly complex. Most likely, testers and developers will want to do many analyses of varying scope and breadth. For this illustration, I will define the "unit of analysis" as users through one semester at VendorBuilt College.

To apply this model, we must choose a subject to analyze. The subject may be an individual or a group - our choice determines the perspective or point of view for the analysis. In the VBC iNetfolio case study, the most obvious choices are student and faculty users of the iNetfolio system. However, administrators, accreditors, potential hiring authorities, and friends and family also have a stake in this activity. For this analysis, let's focus on the students as the subject. In practice, software developers would examine each user perspective.

In CHAT, the object is directly across from the subject. It can be any of the many different things on which a user focuses efforts: raw materials to be transformed into something else; concepts to be learned; or a problem to be solved. The object is the primary focus of the user and carries the purpose, motive, or outcome for the entire activity. For users in this analysis, the object is the portfolio they create using iNetfolio. (For a detailed look at the electronic portfolio object at VBC, readers may view a QuickTime movie posted at http://www.testingeducation.org/wtst5/FiedlerWTST.mov). But what is the motive? The answer to that question (and there could be many) speaks to the outcome(s) or purpose(s) for the activity. From the student perspective, the fact that creating the portfolio is a requirement for graduation is certainly a motive for working on it. For many students, that will be the only motive that matters. Other possible motives include record keeping for the learning and the work they've done; usefulness for job-hunting; and supporting applications for advanced degrees or credentials. Objects and purposes may also evolve as the reasons for working on the object change or the user's understanding of the object evolves.

In CHAT, tools mediate the interaction between subject and object. These tools can be tangible (as in a hammer or a computer), symbolic (language or icon), or psychological (mental models or heuristics). A tool can be used in different ways at different times or across different activity systems. The students in the case study use a wide range of tools including the iNetfolio service, the faculty-created template, computers, scanners, other software, course syllabi, electronic help resources, iNetfolio tech support, and many other things. They even use their professors as tools when they ask them for advice or turn to them for explanations.

Community is located at the base of this model. The community is the group of individuals who share the same object. In the case of students creating electronic portfolios at VBC, the community is comprised of other VBC students, the faculty (yes, they are in more than one area of the model), potential hiring authorities, accreditors, and friends and family.

Notice that the intersection of community and object is labelled division of labour. Labour can be divided both horizontally and vertically. Horizontal division of labour is across members of a community with approximately the same status. In our case study, senior students shared their portfolios with junior students to give ideas about what was expected. They also helped each other think about some of the decisions that went into the portfolios. The vertical division of refers to tasks divided across the different divisions of power. In this case, faculty developed a template to help them with their task.

The final node in this model is at the intersection of community and subject. The interaction of the subject with the community is typically governed by rules, norms, and conventions. These may be formal (part of the system), informal (quirky adaptations by this group) or technical (policies, laws, or mandates). In the VBC iNetfolio case study, rules include the portfolio requirement, the rules governing the format and structure of the portfolio, the methods for turning it in, the fact that it is a graduation requirement, and the custom of senior students sharing portfolios for their younger peers to review.

Networks of Activity

CHAT proponents embrace the networked nature of activity. To capture a robust picture of the central activity and to maximize their understanding of the activity, a CHAT methodologist examines how and when other activities intertwine with the one they're interested in. This usually requires adopting a variety of perspectives. In terms of software testing and development, this helps develop a rich understanding of the software user's context of use.

Engeström identified four kinds of related activities: object-activities; subject-producing activities; rule-producing activities; and tool-producing activities. Object-activities share the same object as the central activity – they are embedded or nested within the central activity. In the case study, Cathy, Tucker, and Rafael told of their need to scan documents (an embedded activity) in order to complete their electronic portfolios (the main activity).

Nearby activities also influence the central activity. Subject-producing activities recruit or train participants in the activity. At VBC, Anne and Alicia were trained by their use of MySpace – influencing their expectations of what should be possible and providing ideas to work around iNetfolio constraints. Students attended classes (one at the beginning of their program and one very near the end) to learn how to use iNetfolio and the rules surrounding its use. Rule-producing activities focus on creating rules, policies, and legislation that impact the central activity. Accreditors and administrators convert legislation into policies that govern activities in higher education institutions. Such policies at VBC required student participation in the portfolio activity as a condition of graduation and resulted in specific rules governing both form and content of the students' electronic portfolios. Finally, tool-producing activities focus on developing tools to be used in the central activity. In the case study, faculty created a template to help students complete their portfolio task; iNetfolio provided technical support; and students provided model portfolios to younger students.

Looking at the network of portfolio activity, it's easy to see that the stakeholders with power and influence are accreditors and administrators. Software development efforts will need to satisfy these powerful decision-makers for their software tools to be adopted. However, to retain existing customers beyond the initial subscription period, students and young professionals will need to see value in voluntarily renewing their subscription.

Tensions Within Activity Systems

The interplay of nearby and embedded activities can result in "tensions" or "conflicts" within the main activity. In his initial work on CHAT, Engeström addressed these "tensions." The presence of tensions is neither inherently good nor bad. It is expected and does not necessarily indicate a dysfunctional system. Tensions often motivate a system to make changes. Some changes will be viewed as beneficial and quickly adopted by others in the system. Other changes will fail. Understanding the tensions a software user faces, and their perceptions of those tensions, can help developers improve their software.

According to Engeström, there are four levels of tensions. Primary tensions occur within a node of the CHAT model. In his interview, Tucker talks about how he can not figure out how to display a grading guide completed by one of his professors and already stored in the iNetfolio system to display in his electronic portfolio. Cathy tries to use a combination of tools – Microsoft Word and iNetfolio to work on her portfolio. In her interview, she talked about how "everything just goes crazy" when she tries to copy a Word document and paste it into the iNetfolio system. Each of these examples illustrates a tension within the Tools node of the VBC portfolio activity in that the tools do not perform as the users expect they will. Secondary tensions are problems that occur between any two nodes of the activity as modeled by CHAT. Tucker's challenge with displaying the grading guide also represents a secondary tension – one between rules and tools – because the tool appears incapable of doing what he needs it to do to satisfy one of his requirements. A third level of tension, or tertiary tension, can exist between one form of an activity and a more advanced form of that activity. In her interview, Cathy explained that Dr. Lancaster does not accept electronic submissions of a major course assignment that must also be included in the students' electronic portfolios. This requirement for a paper submission for the class assignment causes students to spend many hours reformatting those same assignments to later include in their electronic portfolios. This conflict between "the old way" and "the new way" is a typical example of a tertiary tension. The fourth and final level of tension in activity is quaternary. Quaternary tensions exist between the activity of interest and a nearby activity. In the interviews, Mallory talked about how she felt the electronic portfolio she created to satisfy graduation requirements was inappropriate to use as an interview portfolio. Similarly, Anne and Alicia talked about their experiences with the popular MySpace site and their expectations that iNetfolio would offer similar capabilities. Both of these exemplify the concept of quaternary tension – conflict between nearby activities.

Try It For Yourself

This paper introduced you to the CHAT model and a case study of examples. I urge you to conduct your own “thought experiment” by reflecting on your use of software to complete a task. Challenge yourself to thoroughly saturate the nodes of the CHAT model with observations and examples. Notice the embedded and nearby activities that influence your task and how you complete it. Identify the different tensions in your activity of interest and see if you can determine the level of tension to describe each. Adopt a different user perspective of the activity to see what that new analysis reveals. Increase your awareness of the interconnectedness of related activities. Don’t be surprised if you notice the observations you make start slipping around from place to place on the model. That tends to happen in a CHAT analysis as you zoom in to take a closer look at something and then zoom back out for a broader perspective. In the next instalment of this paper, I will introduce you to tools and techniques to help capture and interpret your observations. Cem Kaner will offer his insights about using this model and these techniques for development tasks such as requirements analysis, specification-based testing, and scenario testing (Kaner, 2003).

References

Cohn, E. R. and Hibbitts, B.J. (2004). Beyond the electronic portfolio: A lifetime personal web space. Educause Quarterly 27(4) p. 7-10. Retrieved December 30, 2006 from http://www.educause.edu/ir/library/pdf/EQM0441.pdf

Engeström, Y. (1987). Learning by Expanding: An Activity-Theoretical Approach to Developmental Research. Retrieved December 30, 2006 from http://lchc.ucsd.edu/MCA/Paper/Engestrom/expanding/toc.htm.

Kaner, C. (2003). "The power of 'What If…' and nine ways to fuel your imagination: Cem Kaner on scenario testing." Software Testing and Quality Engineering Magazine 5(5), p. 16-22. Retrieved December 30, 2006 from http://www.kaner.com/pdfs/ScenarioSTQE.pdf.

Kaner, C. (2004). Software testing as a social science, IFIP Working Group 10.4 Meeting on Software Dependability, Siena, Italy. Retrieved December 30, 2006 from http://www.kaner.com/pdfs/ifipkaner.pdf.

Kaner, C. (2006). Software testing as a social science. Canadian Undergraduate Software Engineering Conference (CUSEC), January 19-21, 2006. Montreal, Canada. Retrieved December 30, 2006 from http://www.kane

Interview with Usability Evangelist James Hobart

James Hobart is an internationally recognized User Interface design consultant and president of Classic System Solutions, Inc. He specializes in the design and development of large-scale, high-volume, user interface design. He is an expert in GUI design for transaction processing systems and web and portal application strategies. He speaks regularly about user interface design topics and topics at national conferences. His industry insights are frequently sought by leading publications including PC Week, Information Week, InfoWorld and UNIX Review. He can be reached through his website which is www.classicsys.com..

While he was in Toronto he was interviewed by the magazine about the work he does and the issues around users and usability.

Jim, can we start the interview by you telling us about what you do?

We currently help organizations to achieve higher degrees of usability and increase the user experience for their customers. The way we do it is in a variety of ways. We have enterprise software called GUI Guides, which are enterprise standards and guidelines and include design best practices. We help customers to implement, sell and use these in their organizations.

We offer consulting. We actually work with teams to drive out their real requirements and critically analyze their existing systems. Then we help them design systems that will hopefully increase the users' experience, usability and meet the needs of the users.

We offer training. We train teams so they can be self-empowered to implement usability with a higher level of success on a continuous basis. When we look at the success of organizations we have more of a long term approach. We look at success over a number of years. Rather than just a single project, that's the true determination.

You also do usability testing, can you expand on this?

Yes. We validate our designs and we do usability testing for clients. For projects we are working on, we use our usability lab to validate our design assumptions. I'm humbled every week by the results in the lab and what we learn by studying people at a very detailed level. We try to understand true human behaviour and intentions as opposed to what people tell us that they want. We watch how they behave.

So have you worked on any software projects or companies that we could relate to?

Sure, we have worked on a variety of projects over the years. We have worked with NASA. In the last few minutes before launching a space shuttle there is a lot of data, a lot of information. They worked with our team to create tools so that mission control would actually know when to launch and if everything is okay or not. Lives are at risk. It's a huge program with a high degree of complexity.

We've worked with dot-coms, e-commerce sites, and insurance companies. Progressive Insurance is an example. We've worked with insurance companies to help people apply for auto insurance online. We've worked with a number of call centers including ones in Canada. We worked with high end systems, like flight trainers for F15s and Apache longbow helicopters. Where you know, lives are at risk. Everything from that to money trading systems on Wall Street. I really enjoy working with the high intensity systems, whether money trading or flight systems. The software has to be very well tuned to the needs of the user or it will simply fail. So the things we learn on Wall Street are very applicable to less intense applications else ware in the usability domain

I've been to lots of companies and usability doesn't seem to be a really big issue. So as you watch most companies, you really start to wonder how important usability really is, because so few people are doing it. How important is usability, and why is it important?

Well I think things are changing. So when the senior executives of companies like Apple, Sony and Boeing are realizing you can have brilliant engineers, brilliant hardware and you can have brilliant software, but if the software and hardware don't come together to create a good user experience, an optimal user experience, it really reflects either in revenues or support costs. So what we are seeing is dialogue at a much higher level in organizations. Unfortunately, sometimes that dialogue has a hard time translating into actions at the lower levels of organization, a clear direction.

What we like to avoid is the situation where a company maybe has what I call a "shot of usability." They have one quick initiative and think, “Okay we understand the problem; now what's the next problem?” Usability is a component of quality. Companies that deliver consistently high quality don't say, “Well I worked on quality this month, so therefore it's good.” You're never good enough with quality, and I believe that in usability also it's never good enough. You're never done working on it. So that’s what we are trying to translate. And companies that get it and excel don't treat usability as a one off, it's a constant improving process.

Can you give me an example of a company that didn't get it and it cost them dearly?

Rather than talking about companies that don't get it, I think I can talk historically about a company that maybe was missing the mark and yet it's a world class company but now does get it and really is putting all the guns and effort into it from the CEO down. A world class company Sony. Sony is a brand that is known throughout the world. Historically it has been identified with high quality, tier one products. They have world-class engineers; they dominate in a world-wide marketplace. And yet, a series of things happened over the last few years that really made them look at themselves. Even though they were excelling in many areas, they were probably having some challenges getting usability to the level of quality that was expected of a world class brand, and have that quality appear consistently within all the different lines of business throughout the organization. They actually called us and we were asked to come in and help them. They are using our GUI Guide enterprise design portal. What they found was that they had different lines of business with varying skill levels in design and usability. They had pockets of knowledge that weren’t being shared among the different business groups. They had a lot of re-design efforts and work that was being duplicated across the different organizations. So they weren't getting a significant ROI for the design efforts that were being put in. At the end of the day, the hardware didn't always work as well with the software as it should have, to meet the demands of the consumer. That gave an opportunity for the iPod and iTunes to make a major dent. I think when Sony realized that iPod and iTunes were very successful, and yet here was Sony with a large install base of Walkmans, the largest install base of mobile devices in the world, and the largest collection of music with MGM and CBS records. And yet they had a sizable and dominant competitor in iTunes that came out and wasn't the traditional competitor.

So to make sure this doesn't happen and they continue to dominate in all these fields, they are putting a very strong effort towards usability. They created a user designer center of excellence. The center of excellence is consolidating the best design practices, building pattern libraries and collections, and communicating these design best practices globally throughout the organization. The effects of this will be seen in the upcoming years. I think it's a really smart move, and it's a move that Sony really had to do to protect their brand.

Can you give me some examples of really good usable websites?

Well usable means different things for different people. So an example of a commerce website would be Amazon.com. Many people are shopping there this time of year. They trust the reputation, the shipping polices, the amount of content and quality of content, and the reviews. So Amazon has been able to put together a set of content and services that translates to revenue. Especially in e-commerce, one of the key markers of usability is revenue. If they are driving revenue and making sales, they are doing something right. In the insurance space a company that has been dominant in this space is Progressive Insurance. Because they are so dominant, they can spend more on advertising, like on Google, and they know it will translate to ROI. They have a high degree of certainty that the people who click on that ad will translate to customers, because when they go to the website they'll have a useable experience, a good experience, and choose to become customers. So then they can pay more for advertising and have a positive revenue cycle.

Where in the SDLC (software development life cycle) do you bring in usability?

LITruism

Usability is a component of quality. Companies that deliver consistently high quality don't say, “Well I worked on quality this month, so therefore it's good.” You're never good enough with quality, and I believe that in usability also it's never good enough. You're never done working on it.

This is a trickier issue because many companies have an entrenched SDLC. We have come to 2 observations. In large companies oftentimes you need a matrix, or variations of the SDLC depending on the size of the project. As you incorporate usability, you should not have a separate usability lifecycle. You need to add specific steps and processes into existing SDLCs throughout the entire project. You do this at the requirements stage, user definition and understanding phase, in the development phase and testing phase to insure that aspects of usability are address at these particular areas in the existing SDLC

Is doing usability testing in the testing phase too late?

If that's the only usability testing then yes it is usually. We get called oftentimes to do usability testing because traditionally people think it's something you do with system testing. We believe usability testing should begin with the prototypes. So we validate conceptual models and prototypes with the usability tests. Because of this, we work with organizations to add steps much sooner in the SDLC. When we can test prototypes, it validates our design assumptions, and gives us a much stronger model. In addition, it helps QA build test cases long before the coding is complete.

Who should do usability testing?

If testing departments are going to take this up I think they could do this. They would have to be trained to be observers and to understand good user interface design guidelines. They would have to have the soft skills needed when interacting with test participants. Ask questions in a way that will not guide them. They would need to be trained on how to run usability lab hardware and software. There is a lot of Audio/Visual equipment and a lot complex software. On the other hand, people in QA have a lot of strong skills that map very well to a usability testing group. Attention to detail, ability to put together very detailed and clear test plans. Ability to eliminate variables and consistently test each usability test participant in a consistent fashion. And then to analyze and translate data into findings. These are all things a QA team already does, Usability Testing should be a natural extension of those skills

Where are the issues with usability right now and where do you see them in the future?

Well first of all, the web is changing with the direction of Web 2.0 and Ajax. The web 2.0 phenomenon and Agile Technology are going to be able to give users the best of the desktop and best of the web together. But that creates a much more complex UI and interactive UI. Just in terms of straight web application designs, we have a much more complex environment to test in, just because of the high levels of interactivity. Before Ajax and Web 2.0, after clicking on a form, we had to wait for the form to submit, and then reload the page. Now, when you're interacting within individual fields and forms, you can be making small trips back and forth to the server as you're working. So it's as though you're working on a desktop application.

There are much higher levels of interactivity and state management, where forms or objects become alive on the desktop, and they are changing state as users are performing certain actions. This is an exponentially higher level of complexity that you have to test.

DETROIT CAR SHOW
Welcome to the Detroit Car Show. Please unveil the first car ever built by a team of programmers.

The other thing with Web2.0 is real time collaboration. You are working with others, and you are hooking into web services that are residing on disparate systems somewhere else in the world, many not in the domain of your country. So you are using maps from Google, addresses from some other service, phone numbers from another service and weather from yet another service. As you're pulling in data and integrating with all these other disparate web services, you have to account for things like down time. If you have queued up a request and the system is down, how do you respond?

Now you have many things that are out of your control. And as you are storing data and sharing data, if we are both working on the same items at the same time, how do I know that I am not overwriting your data? Example is Writely, Google's new word-processor. 100 people could be working and typing into the same document all at the same time. No one knows where the document is stored, it's just somewhere on the Google internet data storage system. Google had to write the proprietary file system to manage this complexity, but we are not Google. So in your company, how is that going to work? How are you going to allow the level of collaboration that people have come to expect, and make it reliable and scaleable?

So for testers, the issue would be a new degree of complexity?

Much higher degrees of complexity, not only from the front end, but very disparate and iterative interactions with backends as well. If you are integrating maps, every time I move around a map I'm sending hundreds of requests to a map server on some other disparate system. The maps have to respond and send data back, and respond properly.

What we have seen are performance issues. Maybe before, people had a map server, and only a few people in the organization used that fancy mapping system. Now we've made it available on the Internet, and anyone can use the maps. So now there's a thousand-fold increase in usage of that map server, and no one ever accounted for that volume of activity, so you have performance issues. Those are the kind of things we are seeing.

Beyond that, we are seeing UI applied to a variety of platforms. We are moving away from the desktop to web applications, PDAs, and handheld devices. So in areas like China, the most common UI for internet browsing is a cell phone. We are quickly going to see these mobile devices taking hold. And combinations of different types of interfaces to use these devices. For example, voice input into a phone and responding back with maps. Or GPS built into a PDA or phone, so they can actually provide location based assistance, like the nearest Starbucks. It will tell you make a left, make a right and you'll be at the Starbucks. And it's not too much further for Google to know that you are near the nearest Starbucks and you press a button and it will have your favourite coffee ready, maybe at a discount.

Wouldn't the government know where everyone is at every moment too?

That's the scary part. Personal privacy is an issue with all of this, and will have to be addressed. When these mobile devices are with us all the time, they will have an ability to act like a personal butler.

The other area we work on is the "attentional user interface" with all these things asking for our attention. These devices that are with us will have to know when to bother us with information and when not too. Hopefully, they will do a good job and enhance our lives for the better.

We are talking about the future. To end, could you give us some wise thoughts of what we should be aware of, and what we should take into consideration?

Usability is a relative young field relative to other disciplines like engineering which has been around for thousands of years. We have a tremendous amount still to learn about how humans behave and how they interact. The guidelines are not set in stone, the technology is changing quickly. We've learned that you can never know enough about the end users and what drives their motivations. So whether you can afford a full lab or start off what a portable UI lab, studying true behaviour of your users on your systems is a key component of gaining insight of what will work and what won't work. It will guide your design decisions.

As you get more competencies, it's important to communicate these best practices in design to other parts of the organization. So you should have a plan in place to educate your staff and communicate this consistency across the organization.

One of the best things you can do about usability is from Jeff Brask who is one of the head people at Apple. He always was a proponent of what we call "instilling habits in users". The way we do that is consistency, consistent behaviour. Usability is really making sure that software behaves the way the user expects, and every time it doesn't, we have to understand why, and hopefully adapt the software so that it works the way we expect it. We do that by making good decisions. There are many things that could happen to make bad decisions or no decision; that's when we end up with a thousand options under the option menu. Or a confusion portal with a hundred links on it. So we want to step back and make good decisions and make them faster and get to the right solution successfully and get it right the first time.

There is a move nowadays to off-shoring development and testing. Do you think usability testing will be eventually off-shored?

It's already happening to some extent. There are some things to consider. Firstly, because of the development being offshore, it's much more important to clearly define the requirement, and more important to prototype and usability test before you send your project offshore to be coded. It's a higher reason to usability test sooner rather that later.

As far as usability testing projects offshore, I have a few challenges with that approach. A lot of good usability testing is predicated on the ability of the person administering the test to clearly understand the implications and attitudes and motivations of the user. You aren't just tracking task start and stop times; you are watching very subtle qualitative behaviours of the user, and have to be able to identify them. If the person who is facilitating usability tests can't pick up the qualitative issues, then this affects the overall usefulness of the test. Such things as subtle intonations and subtle facial movement could have a huge impact on the success of the software.

Our belief is that if you are testing software for a specific audience, the tester should be in tune with the cultural aspects of that particular audience. We test a lot in the US, and we feel we have a good understanding of the users we are seeing in the labs. However, we would not be effective testing in people in another culture, where we don't understand the subtle aspects of their culture. People in India, unless they are very in tune with the culture in the US, may not have the same success.

You are talking about things like body language.

Yes. Body language and expressions, they may not pick up on those. A large part of the benefit of the usability test would be lost.

Ideally, this is what I would like to see: If there is a push to lower the cost of usability testing with outsourcing, then my first question is; why are we not demonstrating value as an industry? I would rather we demonstrated more value to the project, and that would justify the work we are doing here as apposed to lowering the cost.

Thank you so much for speaking with us today.

My pleasure.

LITruism

Usability is a relative young field relative to other disciplines like engineering which has been around for thousands of years. We have a tremendous amount still to learn about how humans behave and how they interact. The guidelines are not set in stone, the technology is changing quickly. We've learned that you can never know enough about the end users and what drives their motivations.

I Speak For The User: The Problem of Agency in Software Development

From Cem Kaner

Cem Kaner, J.D., Ph.D., is Professor of Software Engineering at the Florida Institute of Technology. His primary test-related interests are in developing curricular materials for software testing, integrating good software testing practices with agile development, and forming a professional society for software testing. Before joining Florida Tech, Dr. Kaner worked in Silicon Valley for 17 years, doing and managing programming, user interface design, testing, and user documentation. He is the senior author of Lessons Learned In Software Testing with James Bach and Bret Pettichord, Testing Computer Software, 2nd Edition with Jack Falk and Hung Quoc Nguyen, and "Bad Software: What To Do When Software Fails" with David Pels.

Dr. Kaner is also an attorney whose practice is focused on the law of software quality. Dr. Kaner holds a B.A. in Arts & Sciences (Math, Philosophy), a Ph.D. in Experimental Psychology (Human Perception & Performance: Psychophysics), and a J.D. (law degree). Visit Dr. Kaner's web sites: www.kaner.com and www.badsoftware.com.

Many testers see themselves as user advocates and get into passionate arguments over bugs they consider unacceptable to the user. The bugs might be in the code, the visible design, the interoperability with other devices or systems, etc. Other members of the team might disagree with the assessment of the bug ("That's not a bug, it's a feature!") or its priority.

At some point, some testers get incensed or insistent. They feel that the development team has to listen to them because *Testers Speak For The User.*

There are a couple of interesting questions here.

First, testers aren't the only people who have the "*We Speak For The User*" message drummed into them at almost every conference or workshop. Writers hear this a lot too. So do tech support / help desk / field support staff. Usability testers. Installers. System administrators. Marketers. (Yes, really. Marketers.) Salespeople. (Who do the users tell "no, I don't like it and I won't pay for it?" Salespeople! Who has the strongest vested interest in getting it right for the user? Hmmm…)

So when some tester protests in a meeting, "But you have to fix this. The user will hate it. I know the user will hate it because… *I Speak For The User*", what message does that send to all those other people who think "*We Speak For The User*"?

Who's right?

That brings us to the second question. When Person A say that s/he speaks on behalf of person U, that A can decide whether something is acceptable for U or not, and A's decisions are binding on U, then A is speaking as U's *agent*.

How does A become U's agent?

U (or an agent of U) has to appoint A as U's agent. So when the tester, the writer, the tech supporter, the project manager, the marketer, the tinker, the tailor and the candle-stickmaker all say "*I Speak For The User*" the immediate question has to be, "What user has appointed you as their agent?" The next question, of course is, "What about all the other users?"

I have never seen a case in which an authorized user appointed a tester as their agent.

Some users appoint testers as investigators ("get me this information") but I have not seen delegation of authority ("represent me in the meetings and make binding decisions

LITruism

So when the tester, the writer, the tech supporter, the project manager, the marketer, the tinker, the tailor and the candle-stickmaker all say "*I Speak For The User*" the immediate question has to be, "What user has appointed you as their agent?"

on my behalf.")

I think that in reality, the tester has created his or her own private mental model, in good faith, of some subpopulation of the user community and is advocating consistently with that model.

- Most user communities are diverse, with diverse interests and expectations. Even if the tester has accurately modeled one segment of the community, how large and influential is that segment relative to the others? How does the tester know? How accurate are the other models (sales, marketing, project manager, etc.) with respect to the segments that they model and how large and influential are those segments? *The key point to recognize here is that each of the participants in the triage meeting who says, "I know what the user wants and it is …" might be completely correct with respect to the segment of the user community that they have modeled. The natural feeling in this meeting ("I am right and you are wrong") is often inappropriate. The better feeling is ("which groups' interests are we reflecting and how do we get those people to reconcile in this case?")*

- When U is absent, unable to exercise supervisory authority over A, we often see drift—decision-making in which A makes decisions believing they are in the best interests of U, but actually over time, they gradually come to reflect what A wants more than what U wants. When I say "often", I mean this is a standard problem studied in law school courses (that most or all American law schools require) that deal with the law of agency. We study it as a routine situation, one that we will encounter time after time in different variations. Unfortunately, a mental model of a hypothetical person that you hypothetically represent is flexible. It is easy for people of good will who are trying hard to represent interests that they care about to unconsciously adjust their model in self-serving ways. There is no corrective mechanism, no reality check that halts and reverses drift.

So, what does this mean?

Does it mean that testers should never advocate on behalf of users? *It better not, because in many development situations, users have few or no effective representatives. Even in the XP world where there is the designated One True Customer Representative, what segment(s) does s/he represent and who is being ignored? The XP process books that I have read assume that the stakeholders will work out their differences, enabling the One True Customer Representative to be 'The Speaker On Behalf Of the Stakeholder". I have not seen effective guidance in those books for achieving the often-impossible-to-achieve consensus that this assumes.*

So if testers should advocate on behalf of users, how should we do it?

1. If you are a user advocate, it is essential to come to understand the system from the users' points of view. No amount of pair programming or regression test automation can ever teach you this. Relative to the goal of user advocacy, these activities are distractions that can leave you incompetent for your task (user advocacy).

2. If you are a user advocate, don't think of yourself as the user advocate and don't think—unless you have definitive research behind you—that your model of the user(s) is complete or completely accurate. Rather than rejecting other views, try to work with the proponents of specifically differing views to find out what the user community(s) really believes about the disagreement.

3. All facts are friendly. Be an empiricist. In the face of argument, go get facts.

Improving Vendor Quality

From Joe Larizza

Joseph (Joe) Larizza is a Quality Manager, for RBC Dexia Investor Services. He is a member of the Board of Directors for the Toronto Association of Systems and Software Quality and is an Advisor to the Quality Assurance Institute. He is a Certified System Quality Analyst and holds a Bachelor of Arts Degree in Economics.

Vendor Quality - what does it mean to you and your organization? Many of us today are dealing with third parties whether overseas or just around the corner. The software solutions that are delivered to us are key to our overall businesses. How may of you have experienced delays caused by the headaches of third party code? These delays can have many root causes such as poor requirements or a lack of understanding by the third party.

This article will address what we can do specifically around testing to help increase the quality of third party code and reduce unwanted delays. There are many options and opinions regarding the quality of vendor code or lack thereof. Albeit not all vendor code is of poor quality; however it does imply room for improvement and the guidelines suggested here, applied with a little common sense, can make a difference.

This article will focus on structuring pre-acceptance testing, but the work done here has paybacks for the overall quality of the delivered code.

Often the contractual arrangements between the organization and the third party specify a certain level of quality that needs to be delivered. If the code does not meet the specified criteria, remedies are often prescribed. Invoking these remedies is not a preferred option because they may involve time delays, haggling and sometimes a souring of the relationship. The ideal option is that the code is delivered up front with a much higher level of quality and this can be achieved by working together.

To improve the quality the key point when dealing with your vendor is partnership building. Remember, your vendor can only be successful if the relationship develops a partnership or a "win -win" scenario, fostering quality and savings for both parties. Any business relationship which does not build and continue to build a strong partnership will in the long run degrade. Quality concerns will surface and could possibly end the partnership.

Information sharing is the key component to the QA program described to target "testing effectiveness". The basic framework is as follows:

1. Create Pre-acceptance test strategy
2. Review and walk-through the testing strategy
3. Develop test cases and expected results
4. Create execution plan and execute plan
5. Review results - acceptance of criteria
6. Product delivery

Create Pre-acceptance Test Strategy - Joint Effort

Working together with your business partner-vendor to decide on the best approach for validating product upgrades/code enhancements. Take advantage of each others' strengths to determine success criteria. For example, an acceptable pass rate on jointly developed testing scenarios is a way to measure success. Focus your efforts on each others' strengths and expertise. Remember pre-acceptance is your way of saying to the vendor we will not accept the code to do our internal validations until it meets certain specific standards that we jointly agreed upon. It also allows the vendor to make sure that their code can pass many of the tests that your organization will execute before they deliver the code.

Review and walk-through the testing strategy - agree to execute plan

LITruism

Sharing your testing scenarios will reduce the number of defects. If the vendor's testing team has access to these test cases and has the ability to execute them in their testing labs, you can expect less defects.

It may appear redundant; however a detailed walk-through of the jointly developed testing strategy and expected outcomes is recommended. Both parties will bring to the table assumptions or expectations that may not be clearly understood. A thorough review will uncover potential misunderstandings.

In some cases the code delivered are adjustments to an existing third party application. The vendor may have a core application that has been customized for a wide range of clients.

In devising a test strategy in this case many high level issues must be resolved such as:

- What is core functionality and what are the code adjustments?
- Who tests what?
- How do we know that the vendor has tested their core functionality?
- Is there a vendor test plan?
- What are the test cases? Are they documented and available for reviewing?
- How will the vendor prove that the core test cases were executed?
- What test cases will the 3rd party execute to test the code adjustments?
- Are these code adjustments included in regression testing?
- Who decides what are the test cases to test the adjustments?
- Who has the final approval?

Develop test cases and expected results - joint effort

The creation of independent test cases may be necessary due to logistic reasons. However, sharing the testing scenarios will reduce the number of defects. If the vendor's testing team has access to these test cases and has the ability to execute them in their testing labs, you can expect less defects. The added benefit is that the vendor's testing team will also be in a position to highlight potential requirement gaps. In general, testing teams- vendors' understanding of the product (hands on daily) will verify design or product upgrades prior to any code development.

In some cases a more complete pre-acceptance test can be negotiated. Here the vendor software must pass specified tests before the organization accepts the code. This has several advantages.

- Code will be delivered to an expected standard that will allow your organization to start testing without having to deal with defects which should have been found by the vendor.
- The vendor has specific criteria that must be met and therefore can work more efficiently to meet them.
- The amount of politics between the two parties is decreased.

Create execution plan - joint effort

Test planning in advance will reduce the "blame game" when dealing with system defects. Plan defect correction cycles in advance and worst case scenarios. Don't assume the happy path for execution based on the above guidelines. A jointly developed execution plan dealing with tough issues will continue to foster a positive partnership - when the going gets tough.

Issues which need to be resolved vary from general to quite specific. Here are some examples of the type of decisions that are required to be negotiated:

- Where the testing is to be executed, vendor site, offshore or in your organization?
- Who will conduct the testing?
- Will there be representatives from both organizations?
- Can testing be executed remotely?

LITruism

The wonderful thing about the web is that no matter how hard the programmer has worked, no matter how long it took to create the web site, no matter how much it cost or how many resources were used, no matter how certain the designers are that they can control the user, it only takes a fraction of a second to hit Home, or the Back button. The user, for that split second, is King (or Queen).

- Will the vendor perform their tests and then your organization will repeat those test and others before accepting the code?

- What data will be used for testing?

- On what environment will the testing be completed and who is responsible for setting this up?

- What is the turn around time for correcting defects?

- Is it possible to automate these tests and who should do this?

- How do we determine that pre-acceptance testing has been completed and the code is now deemed ready for delivery?

Test automation if possible, would be much more efficient then manual tests. If the automated tests can be built by the vendor they can then be delivered along with the code for your organization's testing requirements. In many cases the test automation delivery will become the basis of your regression testing efforts. Beware of vendors claiming that they have significant test automation and ask for proof to avoid glorified smoke tests.

If the client is providing the test automation, arrange to have access to the code while the vendor is performing integration and system testing. In that way automation can be written before pre-acceptance testing and used for both vendor system testing and PAT.

Executing and Reviewing results - acceptance of criteria

Inspection cycles have the advantage of ensuring quality releases. If expectations are planned in advance and jointly agreed-upon as suggested here, reviewing vendor testing results prior to delivery has many benefits. Defects or design issues can be addressed in the vendor's shop where corrective measure can be taken efficiently and can potentially reduce timeline impacts.

Product delivery

A planned re-test of all or partial pre-acceptance test cases in your environment is advisable if product set-up or hardware cannot be duplicated in your vendor's test lab.

Having a development region for your vendor inside your shop will (with the assumption that this is possible within your environment) also increase the quality of the product. This will allow the vendor to fix defects and test them using your configurations and data.

A well worked out structure and good communication is now the single most important factor to increase the chance for success. Given the global nature of many vendors and the ability of 24 hour development cycles, good communication is required as you may require access to people who are asleep.

Conclusion

The ideal scenario is that both parties work and communicate without barriers or as few barriers as possible. The above framework addresses testing aspects of our vendor relationships. This doesn't mean that implementing this approach will resolve all of your potential quality concerns. However, if your vendor relationship is suffering, this approach can begin to build a business partnership. In general, apply common sense and design a working solution with your vendor if you are suffering from poor vendor quality.

Improving Relationships Between Producers and Users

From Michael Pregmon, Jr.

Dr. Pregmon is currently Executive Vice President of the Quality Assurance Institute with over 40 years of experience in various business functions. His experience focuses on the user side of information systems such as customer/user satisfaction, systems modeling, defining system requirements, building and maintaining systems, and acceptance testing. He received his BS degree in Business Administration from Pennsylvania State University, a MBA from New York Institute of Technology and his Ph.D. from Madison University.

Over the years, he has been an author, consultant, senior corporate executive and adjunct professor in the Graduate School of Business at Nova Southeastern University in Florida. He currently is an adjunct professor at St Petersburg College also in Florida. Dr. Pregmon has presented seminars and training programs at over 65 colleges and universities in the United States and in other countries throughout the world. He is a former member of the editorial review board of Software Development Magazine and a current member of the Software Testing Advisory Board at the University of Washington. He also is an Advisory Board Member and Technical Expert with the World Business Review television program hosted by General Alexander M. Haig, Jr. This is aired on CNBC, BRAVO, ASIA TV, United Airlines, etc.

Mike was also awarded and honoured as a Paul Harris Fellow, which is Rotary Club International's highest award for service to humanity worldwide. He is an active member of the Dr. Phillips Rotary Club in Orlando, Florida.

An evolving phenomenon is developing in the information technology quality assurance professional field in an attempt to vastly improve customer and user service. This effort is also aimed at reducing defects discovered throughout the lifecycle development of a product or service. We are aware that a majority of defects discovered during product development are introduced during the requirements stage of a project.

Who are the producers and who are the users?

This should not require an explanation. But ironically, some confusion often exists in organizations. Perhaps this emanates more specifically from the culture of an organization and its structure than any other aspect.

Nevertheless, the producers are the key individuals who develop the program or provide the service. Sometimes the producer wears a number of "hats" during the development lifecycle. This individual may actually write or help to write and/or define the output requirements. That professional may be the actual developer of the program or service. Further, that person may also be the so-called system analyst or business analyst. Yet in some organizations, each of these activities is performed by a different individual.

Each user, on the other hand, is really a customer. The early quality pioneers and gurus such as Deming, Juran, Feigenbaum, Crosby, Ishikawa, Taguchi, etc. clearly established that a quality organization does not differentiate between a user and customer. These are synonymous and must be viewed as such when it comes to quality and satisfaction. When individuals within an organization "treat" their co-workers (user) as customers, the external users or customers will experience that same high level of service.

Who is responsible for inferior products or programs?

Quite often, IT and/or project managers are not solely responsible for product or program defects. These can be traced directly to the customer or user. And, this most often results from a misunderstanding or the lack of knowledge and/or background by the customer in what IT as a provider can deliver. Users typically can recognize the problem they face that must be resolved. However, they also have difficulty communicating that need to IT.

Recently, during a training seminar, a participant with many years of project management experience revealed that he has found many, if not most, customers do not know what they want. So, he has to "make that call" for them. Obviously, the major challenge resulting from this dilemma is whether the end product will be "fit for use." Will there be total customer satisfaction? This is not an unusual phenomenon today in product development experienced by many IT providers.

On the other hand, IT project managers often do not fully understand the business side. As a result, communication breakdown in the typical communications model between the "sender" and "receiver" results. In these cases, perception, professional experience, knowledge and background often affect a true understanding of the needs of the customer. IT managers may view this from a producer's standpoint rather than the user's point of view.

Many companies are recognizing this challenge. As a result of requests from a number of QAI member companies, the Institute has developed specific programs to assist its members and clients overcome this concern. Further, the IT industry is also recognizing these shortcomings, and companies are now training professionals who can effectively "inter-relate" with both producers (IT) and users (business functions)

The true role of the Business Analyst (BA) requires this individual to wear a different "hat" during the life cycle of a project. Initially during the requirements phase, the BA works closely with the customer and user to insure the customer's needs are clearly defined and being met. As the project progresses, the BA moves to the IT side to work closely with the project manager to insure understanding of the requirements and user needs. As the project further progresses, the BA moves back to the user's side for acceptance, user testing, training and installation.

Additionally, companies are also recognizing the need by heavily recruiting individuals with the specific talents. Professional magazines and other publications currently reflect this growing need. Nevertheless, we also recognize much confusion exists between the roles of the individuals who interface in a project. Obviously, the innate ability to adequately and properly communicate with both participants in the development cycle is of utmost importance. This may seem a trite statement. But most often the professional vernacular and / or terminology familiar to each of these participants may not be the same.

Strategies for consideration

A very important quality of the true professional is to insure that the learning process is constantly nurtured. However, the software quality professional faces constant dilemmas during the project lifecycle in satisfying both the producer and user. Foremost for consideration are the issues of risk assessment and risk management. IT quality professionals either are not properly trained in employing the concepts of risk analysis or feel this activity in the development lifecycle is unimportant. As a result, this consideration is often avoided when decision making occurs.

Whether an organization envisions the need for such specific training is secondary to that of recognizing the dynamics and importance of these two issues for effective project management. Effective and timely communication is always a challenge. However, employing the concepts of risk management in the IT environment for decision making purposes is indeed a rarity. We all recognize that most decisions we make have trade-offs. Yet, we so often fail to use the techniques of risk analysis in this process.

In Summary

The most important facet to remember is that developers, project managers, business analysts, customers, and users working together in the development lifecycle of a product must be in constant contact. Customers and users want as much as they can get for the absolute least cost. And, they want it fast! Developers and project managers want as much time as possible, with little cost constraint, and without any change requests. So, the dilemma persists. Effective communication with the proper use of risk analysis provides both sides with a rational approach to completing a project that satisfies the needs of all participants in the development life cycle.

Uses of Test Automation

Richard Bornet

Richard Bornet has been in the software business for over 20 years. The last ten he has spent running various testing departments and creating innovative approaches to improve testing. He specializes in test automation and is the inventor of Scenario Tester and co-inventor of Ambiguity Checker software. He can be reached at rbornet@eol.ca. or by phone at 416-986-7175

Test Automation is one of my passions. I believe that it is not only a useful part of all software testing efforts, but is central to testing software. Software testing without testing automation is like building shelves without carpentry tools, building cars without robots, or building roads without trucks and road building machines. It can be done and has been done, but automation sure makes it easier. In reality, no one would take you seriously if you argued that a skyscraper could be built with a high degree of quality and within an acceptable time frame, by workers only doing their work manually. If you suggested that the work would be done with no cranes, no bulldozers, no trucks and only using Word and Excel, you would be laughed out of the room. But that is the stand that so many people take when it comes to software testing.

It could be argued that a skyscraper cannot be built without proper automation, but software can. But look at it this way: until automation (cranes, bulldozers etc.) came along, skyscrapers were not possible. Maybe we haven't tried to build the software equivalent of a skyscraper yet. But that is a philosophical discussion for another time.

The problem with test automation is that it is seldom truly successful. Most software testing automation efforts fail – in spite of honest effort and good intentions, the testing ends up reverting to manual. To see why automation efforts fail, read my article, "Ten questions you should ask before you embark on a test automation effort" (*Software Quality and Testing – October, 2005*). Suffice it to say, the main reason is that we don't have skilled enough people doing the right things.

Even when successful, automation is seldom used to its full potential.

This article is about all the different things one can do with test automation. Some of these are obvious; some are seldom thought of or even tried.

Common uses of automation

Smokes

The most frequent use of test automation is running smoke tests. Smoke tests are automated routines which check that the basic application functionality is working. For example: opening all the screens, basic data input such as entering a client, testing some basic functionality such as search screens.

If an application blows up while smokes are running, it is probably not ready for production. Some might say that it is not even ready for testing, and some test departments insist that smokes be run successfully before the code is promoted to test.

Regression

After smokes, the biggest use of test automation is for regression purposes. We test as much of the functionality of the application as we can, so that we can re-run the tests in the future. This is often self-limiting. There are not enough resources and time to test everything and to automate it all. The automated test tools are often pushed to the limit and cannot perform certain important tasks, or the necessary coding expertise is lacking.

Generally, the regression tests test more than the smokes but far less than they should.

Load and Stress Testing

This tests how the application reacts to large volumes of data or high transaction load or large numbers of simultaneous users. It is theoretically possible to do some basic load and stress testing manually, but it is difficult. How does one manually simulate a thousand users entering a query at the same time without a thousand testers? The only way to do a thorough job and to be able to track where problem areas occur is to use automation, whether this means buying specialized tools, or writing specialized routines.

But what about…

Smokes, regression and load testing are the most commonly thought of uses of test automation. But there are other uses for test automation that significantly increase the power of the tester and decrease the time it takes to test. In fact, many of these functions could be used much more than the tasks listed above.

Data Creation – GUI

In order to test, one must have data. In many systems, just building the initial data is a time consuming activity. Take the example of entering a new customer into a large loan system. It may not be a question of two or three screens. I have seen systems where entering a complex customer involved over 200 separate actions and over twenty minutes of typing. And that was before the actual test could be run.

One of the greatest untapped uses of automation is the creation of data. That whole initial data setup can be an automated task; let the automation run the customers (or whatever initial data you need) in for you. You could have ten different types of customers already set up. When you need one, you select it from a list and simply run it in. Not only does automation set it up, but it also ensures that it is entered correctly.

You could specify that you want a list of these customers run. All the different types of customers needed for the next day's testing could be created overnight, ready for the testers when they arrive in the morning.

Data Creation – Database

The above automation enters data through the application front end, same as if a user sat down at the computer and typed it in. Automation can also affect data through the back end. This means putting adding, editing or deleting records in a table, calling database stored procedures etc.

We can have automated routines that perform a myriad of tasks. Here is a small sample:

- Removing data, for example a new customer is entered, then the automation removes that customer from the database
- Changing the date
- Backing up and restoring the database, for example when the testers have a database that they control and restore themselves
- Resetting a baseline for a specific table
- Creating flat files

With the proper interface, this can be as simple as selecting a task and pressing execute.

Data Searches

Sometimes a particular test requires a record (for example a customer or an inventory item) with certain specific characteristics. Although that record could be created from scratch in order to be able to run the test, chances are that such a record already exists and can be used. Testers often use search screens to try to find one which matches their needs. Some testers will write SQL queries to try to find such a record.

This is very repetitive and time consuming, and automation can help. The tester should be able to specify the criteria and how many matching records are needed, then press a button and get a list of records. The query is easy to use, no messing with SQL, and can be saved to reuse again and again.

This becomes even more powerful when hooked in with the automation code that tests the application. The tester can request a specific type of record; the automation finds that record and then does the necessary tests.

One consequence of this approach is that there is no guarantee of specific values. For example, if you ask for a client who has an income between 60 and 80 thousand, then the actual income value may vary from test to test. So how do we test the expected result? With an Expected Results Engine (ERE). This is a technique for calculating the expected results on the fly, rather than having a saved baseline. For more information, see my article ERE: The Expected Results Engine (*Software Quality and Testing - September 2006)*.

SPUNTS (Specific Unit Testing or Specific Functionality Testing)

When we think of test automation, we think of testers. But automation tools and principles can also benefit developers, as they too are testers – they are unit testing their own code.

LITruism

Software testing without testing automation is like building shelves without carpentry tools, building cars without robots, or building roads without trucks and road building machines. In reality, no one would take you seriously if you argued that a skyscraper could be built by workers only doing their work manually. If you suggested that the work would be done with no cranes, no bulldozers, no trucks and only using Word and Excel, you would be laughed out of the room. But that is the stand that so many people take when it comes to software testing.

SPUNTS are automated routines which do some specific testing. A simple example is a little utility into which the tester enters some values, presses a button, then the automation enters the values and then tests the expected results. But this can be made much more sophisticated. The important point is that they are geared to specific functionality.

Here are some real life examples:

- A bug was reported that the application seemed to blow up when users were typing, but no-one was sure what key combination was causing it. The developer wrote a utility to input every possible combination of keys and check for errors. The utility chugged away and found the combination that caused the error; this allowed the bug to be fixed.
- A developer needed to test extreme conditions, for example entering alpha data into numeric fields, pasting huge amounts of text into small text fields etc. This needed to be done on many fields in many screens. The developer wrote a little automation routine to do it.
- A developer needed to test every value in a combo box and its effect on the rest of the screen. The developer wrote a utility to select each one in turn and test the result

These kinds of tests are usually specifically written for an application. Sometimes data may be entered into the application, sometimes in the databases; sometimes a message is constructed or read.

Monitoring systems

There is an annoying little light that goes on in my van to let me know when something is not right. This is definitely an automated procedure. Why not build something like this into production systems?

There are companies that make a living from add-on products that test your application as it runs in production. A search on the internet for such terms as “performance testing” or “load testing” will yield many hits. However, many of the same monitoring functions can be built into the application as it is being developed.

Monitoring can have many faces. Automation can test the speed of code or servers. This can be very important in heavily traveled web-sites where a slight delay can have significant repercussions. Expected results can be tested with an Expected Results Engine. Data can be validated. Automation can be set-up to run dummy data into production systems, this data is then ignored by the production system or purged by a batch job. This last type of monitoring can then be used to test changes in the system at System Testing and UAT levels.

Conclusion

As I said at the beginning, test automation is one of my passions. I truly believe that limiting automation to Smokes and Regression is really just not taking advantage of its full potential. I hope, in this article, that I have given you some idea of where we can take automation and given you something to think about. Here’s to better tested software.

LITruism

Why test automation?

Most testers have become excellent kayakers. They can handle many different types of situations. They can go down rapids, travel across large lakes, right themselves when they are upside down, and have incredible maneuverability and flexibility.

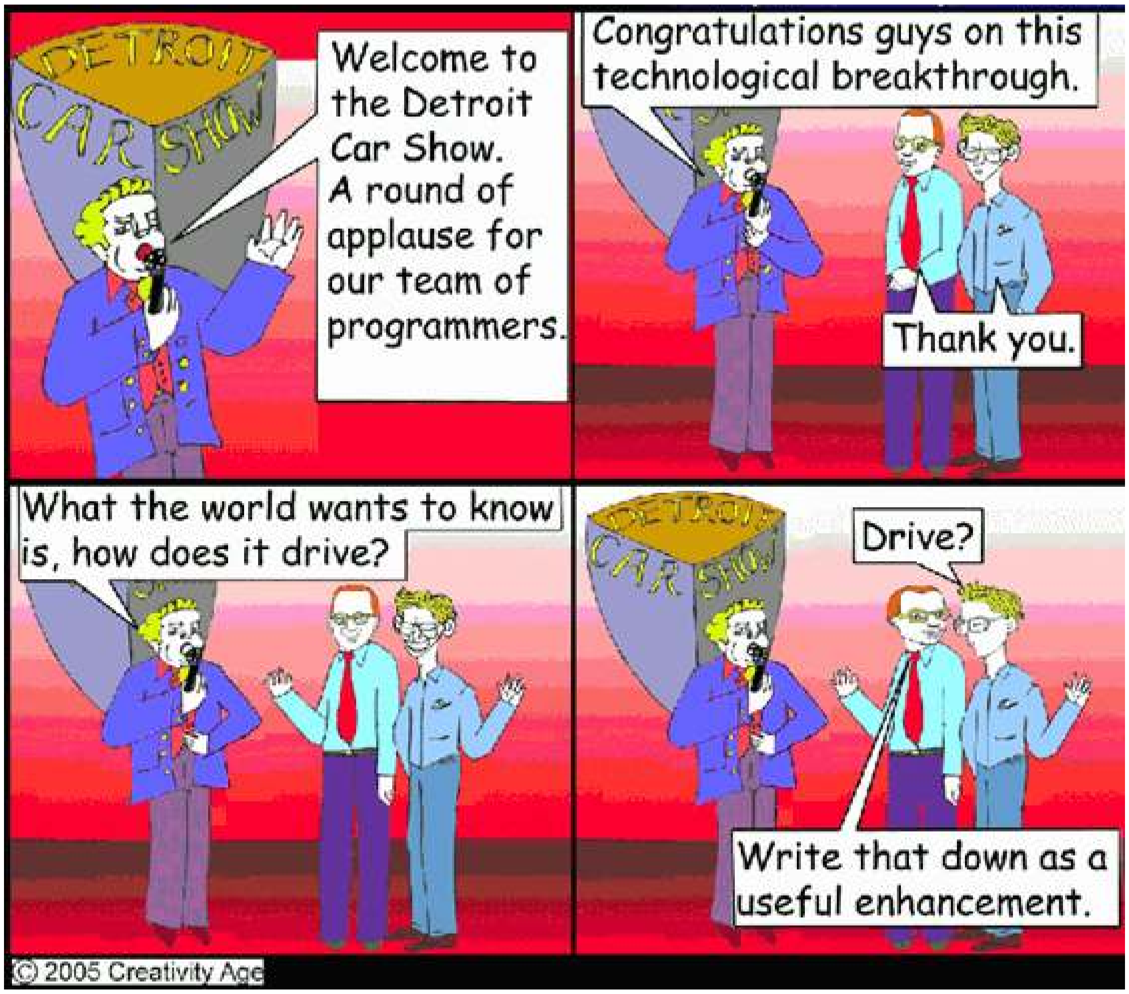
The problem has now become how fast can they get across a large lake, because they may need to do it 30 times a day. Here, at minimum, they would need a very fast motor boat. But as software becomes more complex, they will eventually need to cross the Atlantic, and no matter how many kayakers you have, it is not going to happen. But you could put all those kayakers in a plane and be across in hours.

Events, Conferences, Education and Services

Conference Watch: Here are some upcoming conferences:

STAREast
May 14-18, 2007
Orlando, FL
http://www.sqe.com/stareast/

QAI Canada: Quality Through Global Community
October 1-5, 2007
The Sheraton Centre Hotel, Toronto
http://www.qaicanada.org/conferences/index.html


DETROIT CAR SHOW
Welcome to the Detroit Car Show. A round of applause for our team of programmers.
Congratulations guys on this technological breakthrough.
Thank you.
What the world wants to know is, how does it drive?
DETROIT CAR SHOW
Drive?
Write that down as a useful enhancement.
© 2005 Creativity Age