Exploratory Test Design

DevCon 2011-09-14

Rikard Edgren

TIBCO Spotfire

rikard.edgren@thetesteye.com

This presentation is about how to design tests in an exploratory fashion; aiming to learn more, without stifling execution freedom.

It contains a sub-set of test design heuristics, my current favorites.

Version 1.0

"Exploratory software testing is a style of software testing that emphasizes the personal freedom and responsibility of the individual tester to continually optimize the value of her work by treating test-related learning, test design, test execution, and test result interpretation as mutually supportive activities that run in parallel throughout the project."

[Cem Kaner]

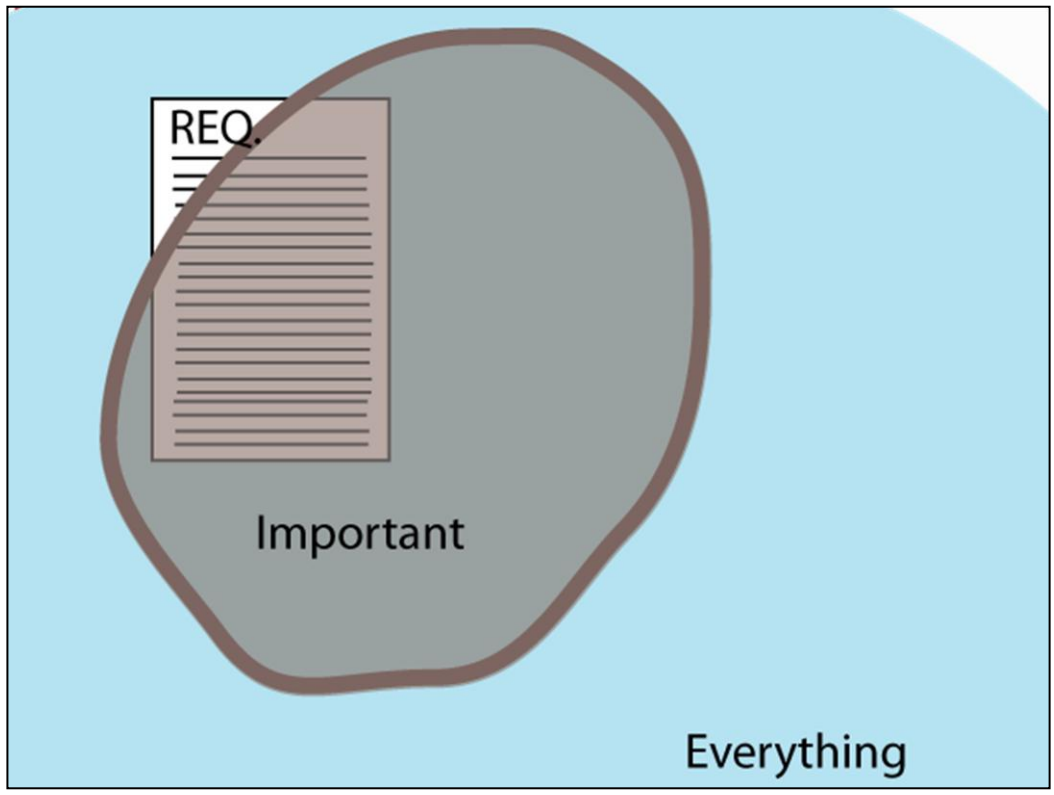
Why is this good?

at start we don't know everything important

we want to know more

we will design tests and learn as we go

This presentation is about designing tests for this reality, where we don't know exactly what we are looking for.



The software potato:

The square symbolizes the features and bugs you will find with test cases stemming from requirements (that can't and shouldn't be complete)

The blue area is every possible usage, including things that maybe no customers would consider a problem.

The brown area is what is important, there lies those problems you'd want to find and fix.

This problem has been solved many times at many places with many different approaches.
What is common could be that testers learn a lot of things from many different sources, combine things, look at many places, think critically and design tests (in advance or on-the-fly) that will cover the important areas.

Some part luck, and a large portion of hard work is needed. Serendipity is working to our advantage.

Recently I realized there can be more than one potato, that are three-dimensional and perhaps slippery; there might be small potatoes that are the best of them'all…

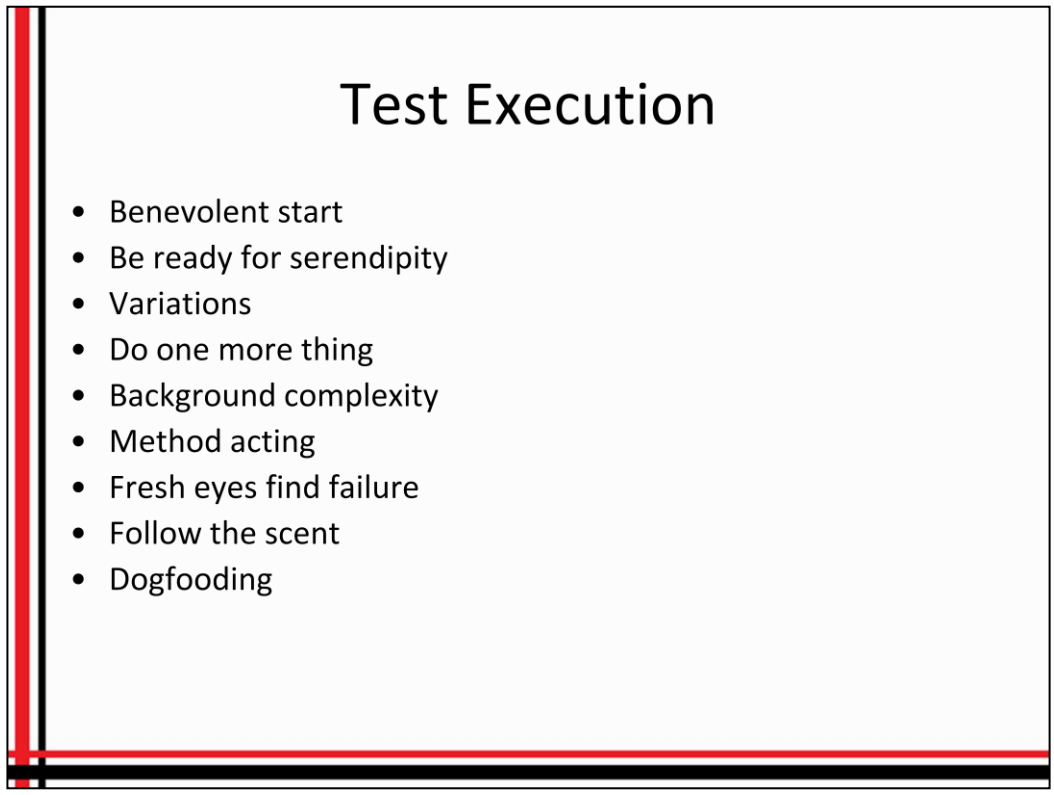
Agenda

1. Test Execution
   - Quicktests
2. Test Design
   - Quality Characteristics
3. Test Analysis
   - Sources for Test Ideas

In medias res...

You design tests all the time.

These activities are sometimes separated, but often intertwined.



Start nice and easy with big chunks, so you find major, major problems at once. Try to do what is intended, before intentionally trying to break it.

At all times, be ready to find valuable things you didn't know you were looking for.

Make Variations all the time, broaden your usage with different data, different paths, so you'll see and learn more.

Whatever test you do, add one more thing, something fast, popular, or error-prone. You might find something important, or learn something.

If you use unnecessarily complex data, you might stumble on important issue. Don't worry that it will be difficult to pinpoint issues, that's better than not knowing about them at all.

Try to be a user, try to mimic their needs, feelings, data and environments.

Look at new things, let others look at your things: fresh eyes find failure

When you notice something odd or strange, investigate at once, or let it rest until you get more clues.

If you can use the software for real, you'll get a better understanding of what and how it should behave. If you're confident of your product, call this "sip our own champagne".

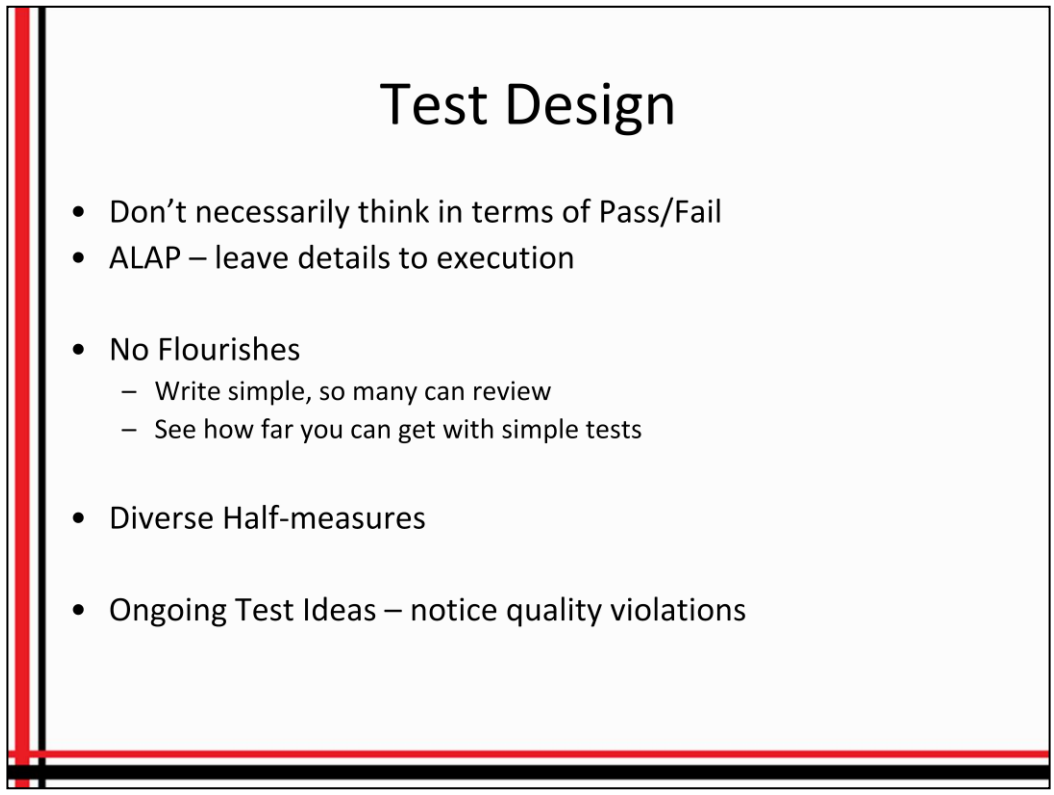
Quicktests

- Continuous Use – leave things over weekend
- Interference – remove, rename, cancel, swap, null
- Data Relationships
- Error-Prone Machine
- Coverage For Free
- Use checklists, your own or
  - Hendrickson, Test Heuristics Cheat Sheet
  - Hunter, You Are Not Done Yet
  - Bach, SFDPOT, HICCUPPSF, CIDTESTD, FDSFSCURA

Hendricksson, Test Heuristics Cheat Sheet, http://testobsessed.com/wp-content/uploads/2007/02/testheuristicscheatsheetv1.pdf

Hunter, You Are Not Done Yet, http://www.thebraidytester.com/downloads/YouAreNotDoneYet.pdf

Bach/Bolton, Rapid Software Testing course, samt http://www.satisfice.com/blog/wp-content/uploads/2009/10/et-dynamics22.pdf



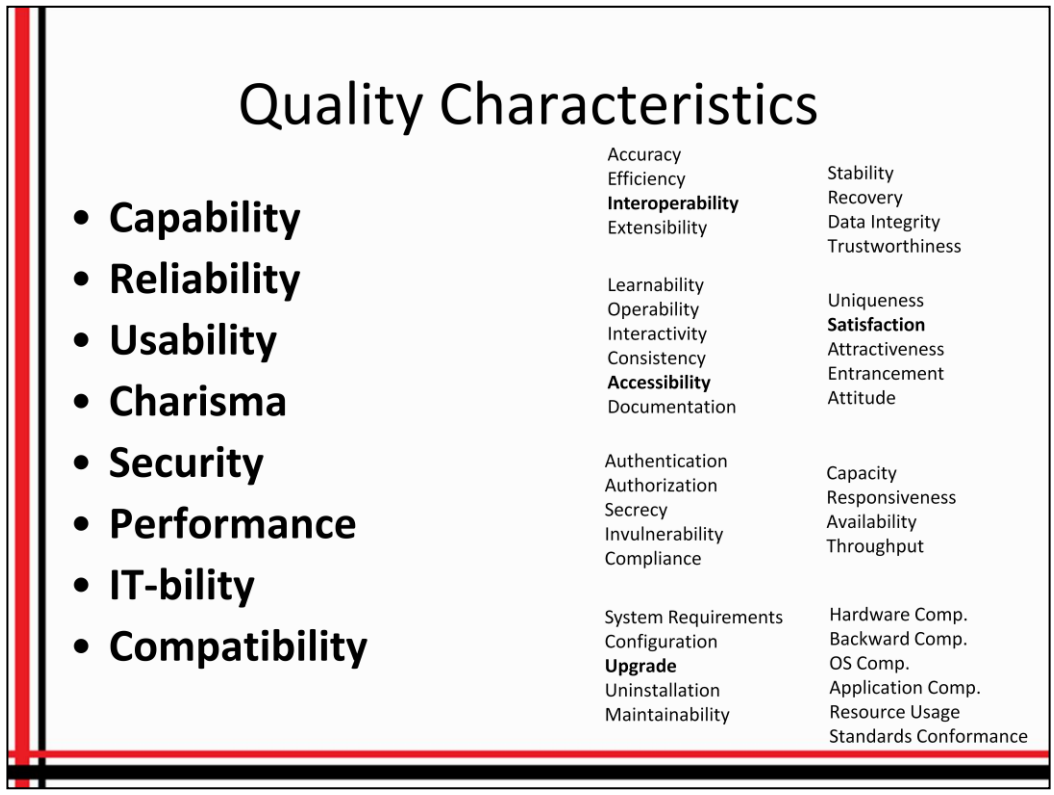
Diverse Half-Measures – "it's better to do more different kinds of testing to a pretty good level, than to do one or two kinds of testing perfectly." Lesson 283 i Cem Kaner, James Bach, Bret Pettichord, Lessons Learned in Software Testing

Ongoing test ideas are used by most manual testers, but seldom talked about.

For example, you'll notice and communicate if a dialog takes 15 seconds to display. (Performance)

You'll get annoyed if something is overly cumbersome to do (Usability, Operability)

You can improve this skill by learning more about Quality Characteristics, and which matters in your context.



Quality attributes are important for all products, not all of them, but many of them.

Capability is the functionality, probably covered in other ways; but interoperabilities are difficult.

Reliability is about handling many situations, and recover from errors.

Usability is also the efficiency you as a tester want to have the 100th time you're using the product. Light Accessibility Testing is very cheap.

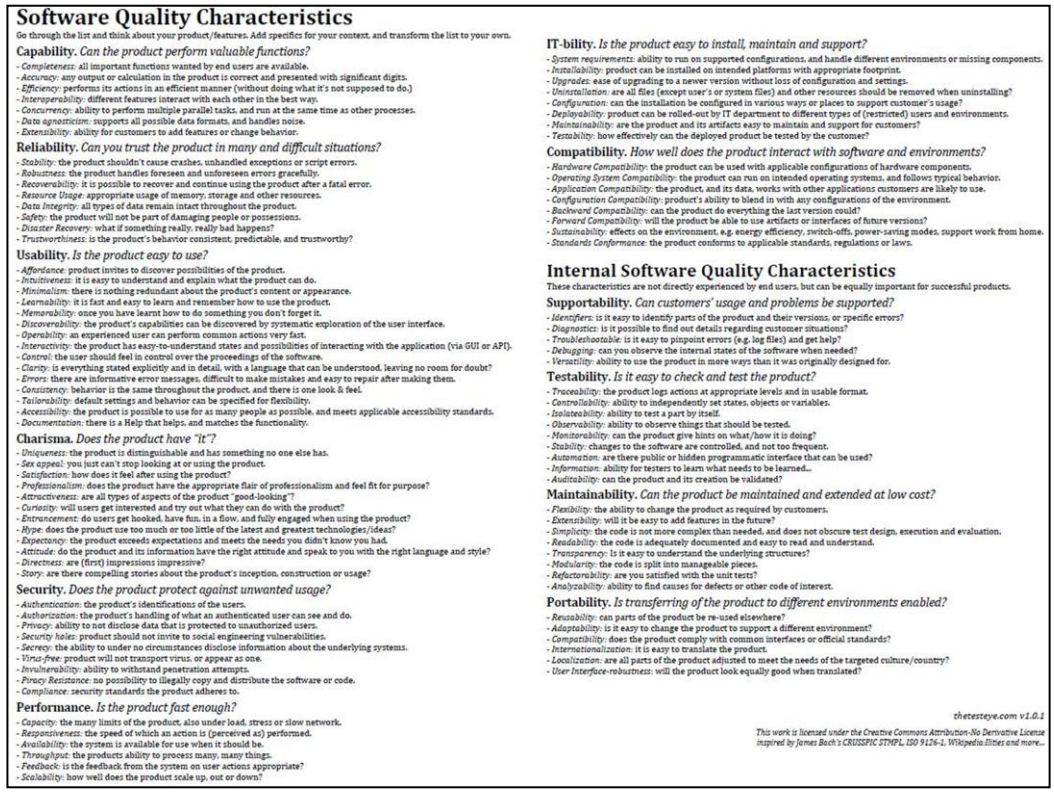
Charisma is important, but seldom tested; does the product have "it"?

Security might seem difficult, but some basics for your specific product you already know.

Performance is both on large-scale, but also for each function and environment.

Installability, Upgrade and Uninstallation; but also that customer IT department can support and maintain the system and its artefacts.

Compatibility with hardware, operating system, software, previous versions, and standards.



Software Quality Characteristics

Go through the list and think about your product/features. Add specifics for your context, and transform the list to your own.

Capability. *Can the product perform valuable functions?*

- *Completeness*: all important functions wanted by end users are available.
- *Accuracy*: any output or calculation in the product is correct and presented with significant digits.
- *Efficiency*: performs its actions in an efficient manner (without doing what it's not supposed to do.)
- *Interoperability*: different features interact with each other in the best way.
- *Concurrency*: ability to perform multiple parallel tasks, and run at the same time as other processes.
- *Data agnosticism*: supports all possible data formats, and handles noise.
- *Extensibility*: ability for customers to add features or change behavior.

Reliability. *Can you trust the product in many and difficult situations?*

- *Stability*: the product shouldn't cause crashes, unhandled exceptions or script errors.
- *Robustness*: the product handles foreseen and unforeseen errors gracefully.
- *Recoverability*: it is possible to recover and continue using the product after a fatal error.
- *Resource Usage*: appropriate usage of memory, storage and other resources.
- *Data Integrity*: all types of data remain intact throughout the product.
- *Safety*: the product will not be part of damaging people or possessions.
- *Disaster Recovery*: what if something really, really bad happens?
- *Trustworthiness*: is the product's behavior consistent, predictable, and trustworthy?

Usability. *Is the product easy to use?*

- *Affordance*: product invites to discover possibilities of the product.
- *Intuitiveness*: it is easy to understand and explain what the product can do.
- *Minimalism*: there is nothing redundant about the product's content or appearance.
- *Learnability*: it is fast and easy to learn and remember how to use the product.
- *Memorability*: once you have learnt how to do something you don't forget it.
- *Discoverability*: the product's capabilities can be discovered by systematic exploration of the user interface.
- *Operability*: an experienced user can perform common actions very fast.
- *Interactivity*: the product has easy-to-understand states and possibilities of interacting with the application (via GUI or API).
- *Control*: the user should feel in control over the proceedings of the software.
- *Clarity*: is everything stated explicitly and in detail, with a language that can be understood, leaving no room for doubt?
- *Errors*: there are informative error messages, difficult to make mistakes and easy to repair after making them.
- *Consistency*: behavior is the same throughout the product, and there is one look & feel.
- *Tailorability*: default settings and behavior can be specified for flexibility.
- *Accessibility*: the product is possible to use for as many people as possible, and meets applicable accessibility standards.
- *Documentation*: there is a Help that helps, and matches the functionality.

Charisma. *Does the product have "it"?*

- *Uniqueness*: the product is distinguishable and has something no one else has.
- *Sex appeal*: you just can't stop looking at or using the product.
- *Satisfaction*: how does it feel after using the product?
- *Professionalism*: does the product have the appropriate flair of professionalism and feel fit for purpose?
- *Attractiveness*: are all types of aspects of the product "good-looking"?
- *Curiosity*: will users get interested and try out what they can do with the product?
- *Entrancement*: do users get hooked, have fun, in a flow, and fully engaged when using the product?
- *Hype*: does the product use too much or too little of the latest and greatest technologies/ideas?
- *Expectancy*: the product exceeds expectations and meets the needs you didn't know you had.
- *Attitude*: do the product and its information have the right attitude and speak to you with the right language and style?
- *Directness*: are (first) impressions impressive?
- *Story*: are there compelling stories about the product's inception, construction or usage?

Security. *Does the product protect against unwanted usage?*

- *Authentication*: the product's identifications of the users.
- *Authorization*: the product's handling of what an authenticated user can see and do.
- *Privacy*: ability to not disclose data that is protected to unauthorized users.
- *Security holes*: product should not invite to social engineering vulnerabilities.
- *Secrecy*: the ability to under no circumstances disclose information about the underlying systems.
- *Virus-free*: product will not transport virus, or appear as one.
- *Invulnerability*: ability to withstand penetration attempts.
- *Piracy Resistance*: no possibility to illegally copy and distribute the software or code.
- *Compliance*: security standards the product adheres to.

Performance. *Is the product fast enough?*

- *Capacity*: the many limits of the product, also under load, stress or slow network.
- *Responsiveness*: the speed of which an action is (perceived as) performed.
- *Availability*: the system is available for use when it should be.
- *Throughput*: the products ability to process many, many things.
- *Feedback*: is the feedback from the system on user actions appropriate?
- *Scalability*: how well does the product scale up, out or down?

IT-bility. *Is the product easy to install, maintain and support?*

- *System requirements*: ability to run on supported configurations, and handle different environments or missing components.
- *Installability*: product can be installed on intended platforms with appropriate footprint.
- *Upgrades*: ease of upgrading to a newer version without loss of configuration and settings.
- *Uninstallation*: are all files (except user's or system files) and other resources should be removed when uninstalling?
- *Configuration*: can the installation be configured in various ways or places to support customer's usage?
- *Deployability*: product can be rolled-out by IT department to different types of (restricted) users and environments.
- *Maintainability*: are the product and its artifacts easy to maintain and support for customers?
- *Testability*: how effectively can the deployed product be tested by the customer?

Compatibility. *How well does the product interact with software and environments?*

- *Hardware Compatibility*: the product can be used with applicable configurations of hardware components.
- *Operating System Compatibility*: the product can run on intended operating systems, and follows typical behavior.
- *Application Compatibility*: the product, and its data, works with other applications customers are likely to use.
- *Configuration Compatibility*: product's ability to blend in with any configurations of the environment.
- *Backward Compatibility*: can the product do everything the last version could?
- *Forward Compatibility*: will the product be able to use artifacts or interfaces of future versions?
- *Sustainability*: effects on the environment, e.g. energy efficiency, switch-offs, power-saving modes, support work from home.
- *Standards Conformance*: the product conforms to applicable standards, regulations or laws.

Internal Software Quality Characteristics

These characteristics are not directly experienced by end users, but can be equally important for successful products.

Supportability. *Can customers' usage and problems be supported?*

- *Identifiers*: is it easy to identify parts of the product and their versions, or specific errors?
- *Diagnostics*: is it possible to find out details regarding customer situations?
- *Troubleshootable*: is it easy to pinpoint errors (e.g. log files) and get help?
- *Debugging*: can you observe the internal states of the software when needed?
- *Versatility*: ability to use the product in more ways than it was originally designed for.

Testability. *Is it easy to check and test the product?*

- *Traceability*: the product logs actions at appropriate levels and in usable format.
- *Controllability*: ability to independently set states, objects or variables.
- *Isolateability*: ability to test a part by itself.
- *Observability*: ability to observe things that should be tested.
- *Monitorability*: can the product give hints on what/how it is doing?
- *Stability*: changes to the software are controlled, and not too frequent.
- *Automation*: are there public or hidden programmatic interface that can be used?
- *Information*: ability for testers to learn what needs to be learned...
- *Auditability*: can the product and its creation be validated?

Maintainability. *Can the product be maintained and extended at low cost?*

- *Flexibility*: the ability to change the product as required by customers.
- *Extensibility*: will it be easy to add features in the future?
- *Simplicity*: the code is not more complex than needed, and does not obscure test design, execution and evaluation.
- *Readability*: the code is adequately documented and easy to read and understand.
- *Transparency*: Is it easy to understand the underlying structures?
- *Modularity*: the code is split into manageable pieces.
- *Refactorability*: are you satisfied with the unit tests?
- *Analyzability*: ability to find causes for defects or other code of interest.

Portability. *Is transferring of the product to different environments enabled?*

- *Reusability*: can parts of the product be re-used elsewhere?
- *Adaptability*: is it easy to change the product to support a different environment?
- *Compatibility*: does the product comply with common interfaces or official standards?
- *Internationalization*: it is easy to translate the product.
- *Localization*: are all parts of the product adjusted to meet the needs of the targeted culture/country?
- *User Interface-robustness*: will the product look equally good when translated?

thetesteye.com v1.0.1

Quality characteristics describe attributes that most software benefit from. They can be used on the whole product or for details.

The whole is made by the details. The quality of a detail is defined by the whole.

This is a thorough extension in the same spirit as Bach's CRUSSPIC STMPL.

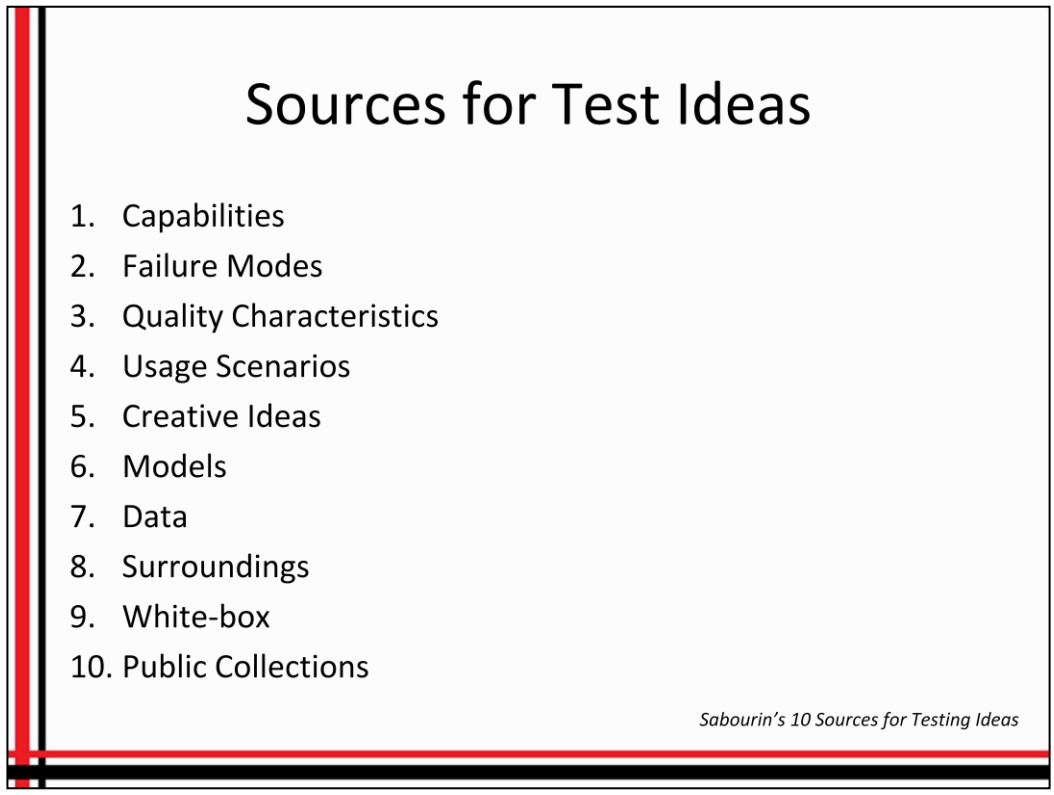
Version 1.0 available at
http://thetesteye.com/posters/TheTestEye_SoftwareQualityCharacteristics.pdf

Test Analysis

- It's about finding out what is important
- Exploratory Test Design looks anywhere
- Mary Had a Little Lamb Heuristic
- Change granularity
- Question everything, especially this statement
- There are many information sources…

Mary had a little lamb" heuristic – accentuate different words, change to synonyms or antonyms. Add "unless", or "...but without"

Change granularity – different levels of details give new perspectives



Each item promotes a way of thinking that can give you good test ideas.

Some of these might not render any test ideas, but they can be very important to have in the back of your head.

The information sources are most useful when they are combined.

What I'm trying to tell you is that in order to do great testing, you must understand what is important, and you do that from a variety of information sources.

Never stop looking and learning.

This list will be released as a poster at
http://thetesteye.com/posters/TheTestEye_SourcesforTestIdeas.pdf

The potato is now filled with content from diverse sources.

Finale

- Drawbacks?
  - Difficult to stop
  - Requires trusting testers
  - Questioned by old-fashioned reviewers
- You have to find YOUR best ways
- Do your best, collaborate, learn to understand **what is important**

http://thetesteye.com/blog/

It is better to stop on purpose, than because you didn't have any ideas.

If you don't trust testers, train them, so you can.

I believe that in 10 years a medical audit might render questions like "So are you really saying you don't do any exploratory testing to find out things you couldn't predict??"

And there are no rules, and the few there are, you should sometimes break.

Questions

- ???

- Further reading:
  - Exploratory Testing Dynamics (Bach, Bach, Bolton)
  - Styles of Exploration (Kaner, Johnson)
  - BBST Exploratory Testing (Kaner, Bach)
  - The Little Black Book on Test Design (Edgren)

Exploratory Testing Dynamics: http://www.satisfice.com/blog/wp-content/uploads/2009/10/et-dynamics22.pdf

Styles of Exploration: http://www.testingeducation.org/BBST/testdesign/KanerJohnson_LAWST7StylesOfExploration.pdf

BBST Exploratory Testing: http://www.testingeducation.org/BBST/exploratory/BBSTExploring.pdf

The Little Black Book on Test Design: http://www.thetesteye.com/papers/TheLittleBlackBookOnTestDesign.pdf