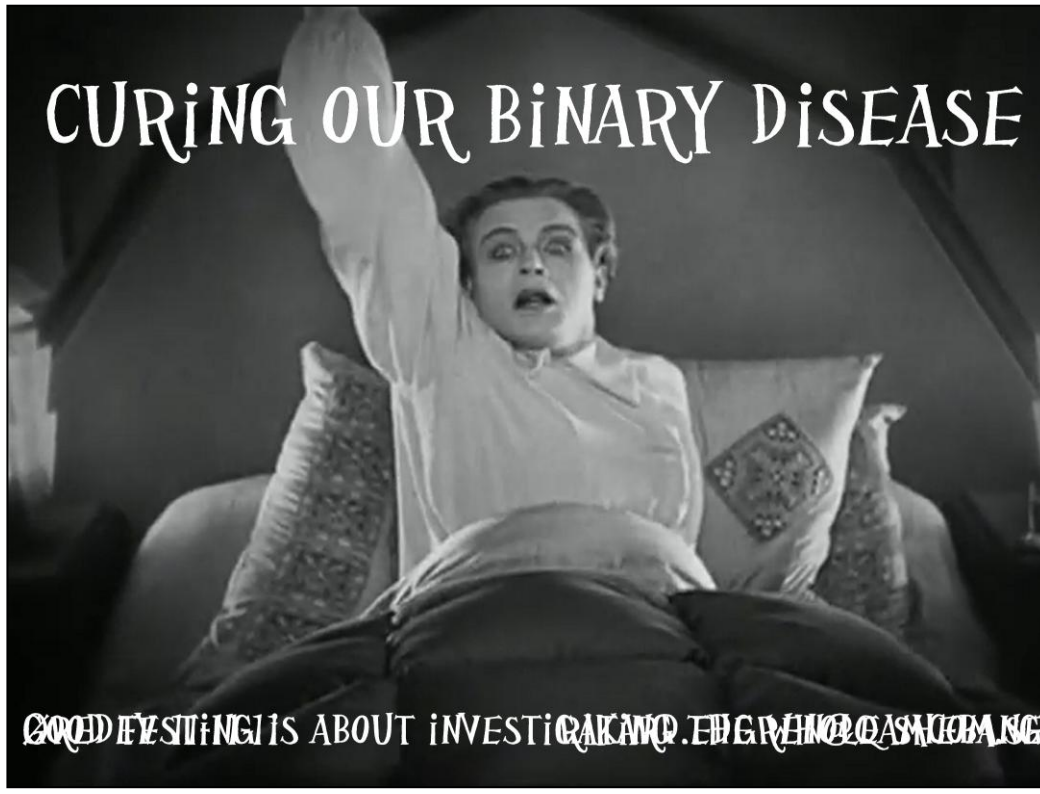
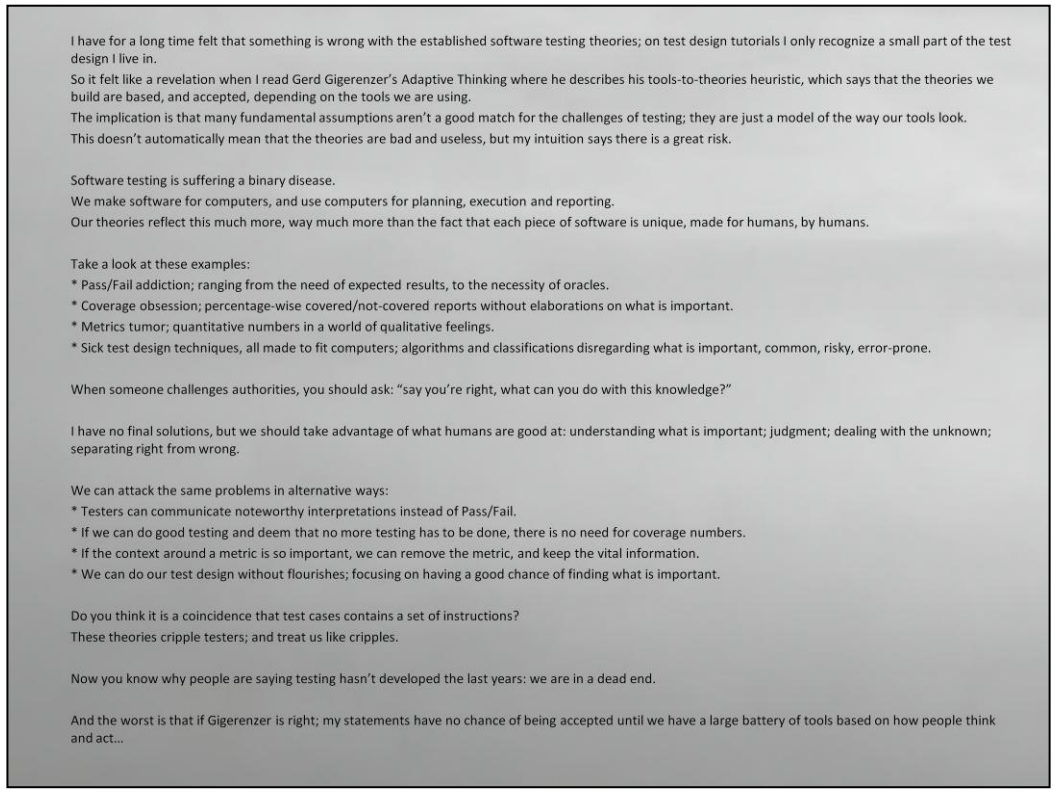


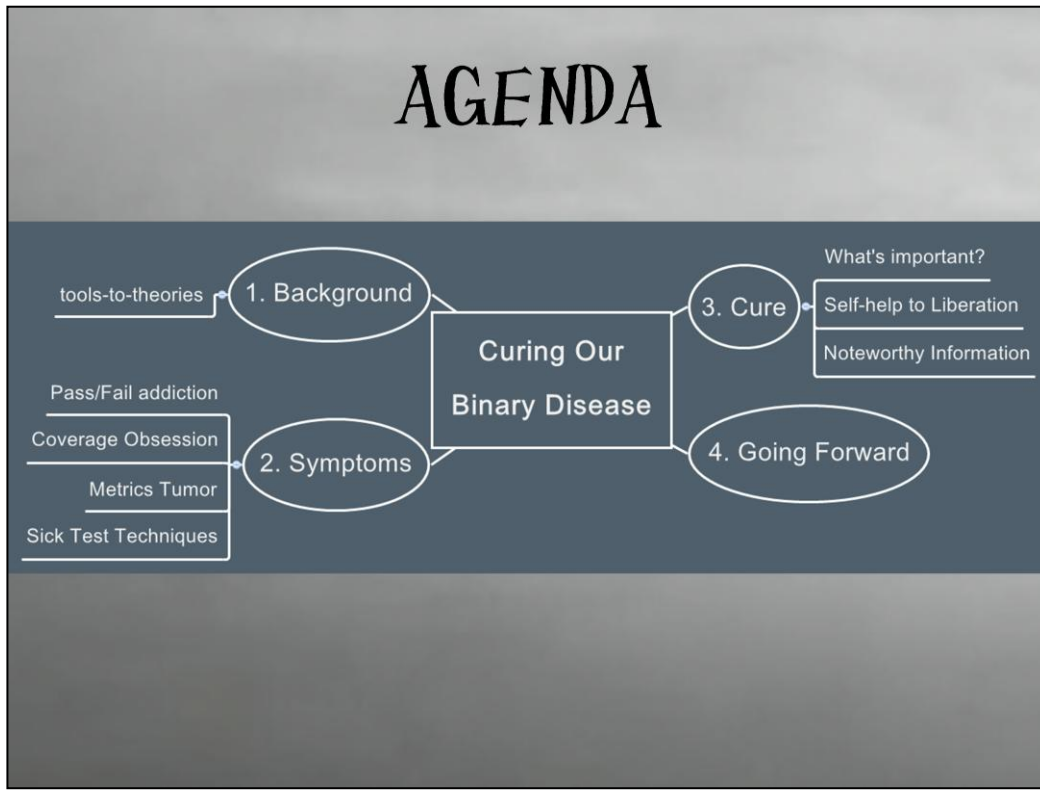
The introduction film was made three years ago, and it isn't about the future, it is about now.

The industry believes testing is supposed to be about binary verification of stories, requirements, examples et.al.



I wrote this abstract for EuroSTAR 2011 and felt extremely pleased.
I had something new and important to say, but it wasn't accepted.
So it became a lightning talk at SWET2, and a blog post, without an impact, as far as I know.
Finally I got to present it here in Malmö, maybe things will start happening now…

I might be very wrong, but I know I'm not.
You might be ill, and I might cure you.



My main message is that if you let go of these crippling computeresque ideas, you have time to focus on what is important.

You might think I'm attacking you, and you might think this has nothing to do with you.

Either way, think about your thinking, think about others thinking, and decide if there is a need for change or not.

TOOLS-TO-THEORIES

Tools shape our theories
Tools shape acceptance of theories

Software testing is a lot about computers
Most software is made for people...

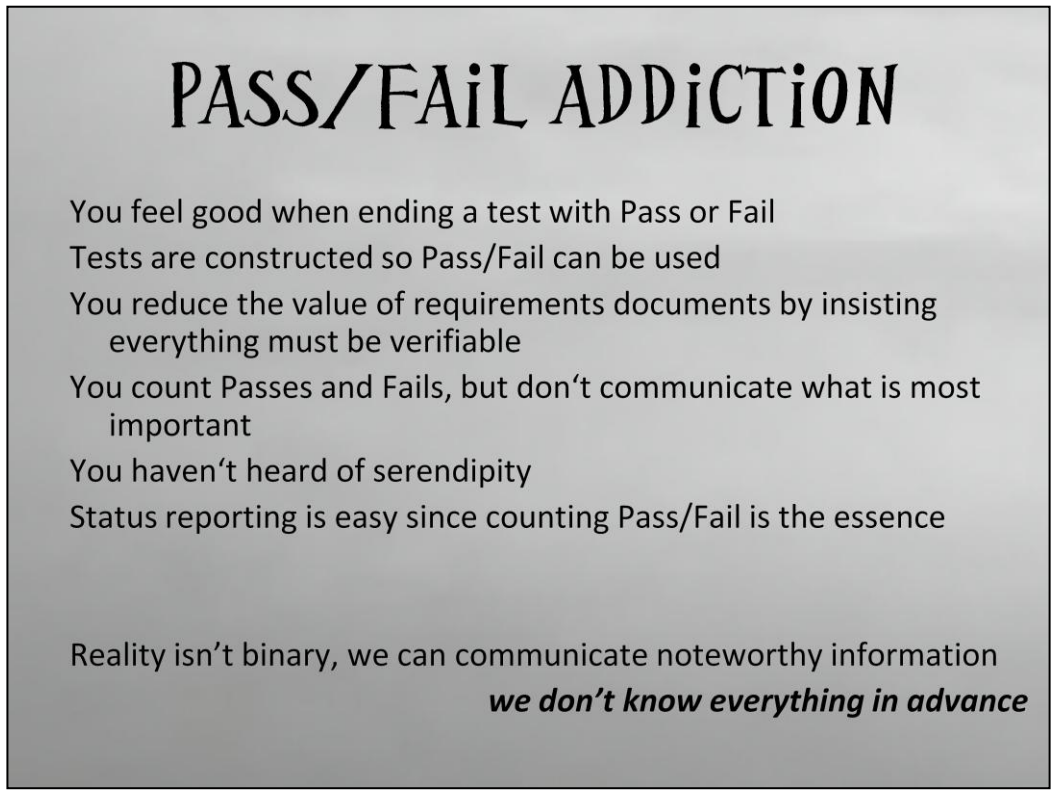
Our theories are way too computeresque

They don't capture what's important

Gerd Gigerenzer

It was a very strong feeling when I read Gigerenzer's tools-to-theories (I found it via Gut Feelings) A lot of the testing theory have felt wrong, and know I could explain why.

This theory doesn't automatically mean that all testing theories are bad and useless, but my intuition says it is more than a risk.



Have you ever seen a test case management systems without mandatory Pass/Fail?
Why do people do all of this pass/fail and metrics on it? Because that's the way you do it.

If we know everything in advance, then Pass/Fail might be OK.
But we never know everything in advance.

You can ask richer questions than is this correct or not? You can learn things, and grow as tester.

PASS/FAIL REHAB

Do some deviations when executing tests

Look at some more places than what is stated in the Expected Results field

Write the occasional test idea using the word "investigate"

Put the numbers in smaller font in your status report

Observe the software without a hypothesis to falsify

You can ask richer questions than: *Is this correct or not?* You can learn things, and grow as tester.

See it as your daily medicine; eventually any Pass/Fail usage will seem ridiculous

Have you ever seen a test case management systems without mandatory Pass/Fail?

COVERAGE OBSESSION

50% coverage can mean

* we have found so many serious bugs that further testing is pointless
* we are running late because testers insist on investigating things they aren't explicitly told to look for
* we have run the 50 most difficult test ideas, and we believe we will finish on schedule
* we have run the 50 easy tests on input data, and look forward to the results from the radically different test ideas
* we have run the first half, in alphabetical order, and are not really sure what we are doing
* we have investigated the 50 most important test ideas, and believe the implicit coverage is enough to go Beta
* we are halfway through, but have found a lot of things that are more important to test than our original assumptions

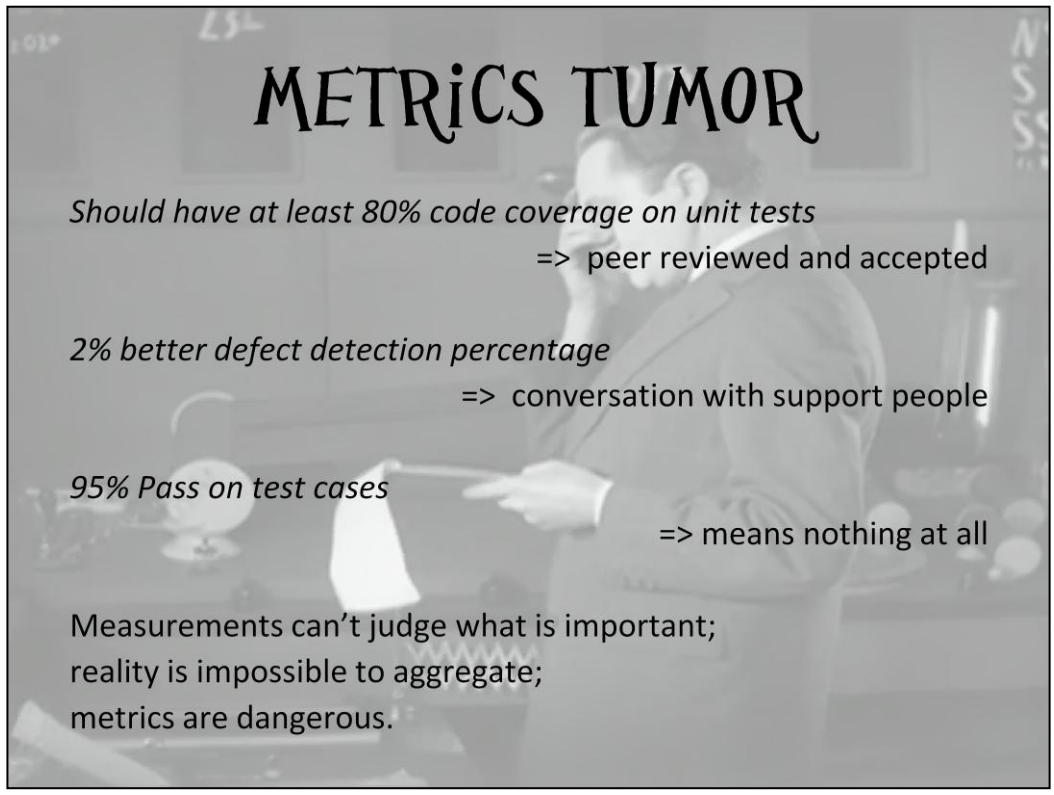
COVERAGE OBSESSION

A coverage model is useful to get ideas

Not useful as a metric of completion

A model can help you find important things, but a percentage number might not include things that are important

Information about the system is more important than information about the model of the system (Emilsson)



At a few occasions I have had the chance to have "the talk" with testers I respect that advocate metrics.

It boils down to the same essence, you need A LOT of context for the numbers to be valid.

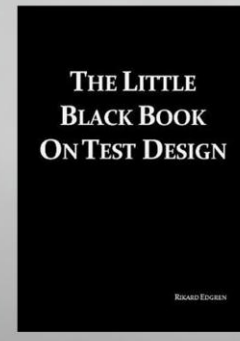
So much context, that I believe you could throw away the numbers and just keep the text.

SICK TEST DESIGN TECHNIQUES

The techniques that usually are taught are old, they are based in computer science and ideas about everything being known in advance

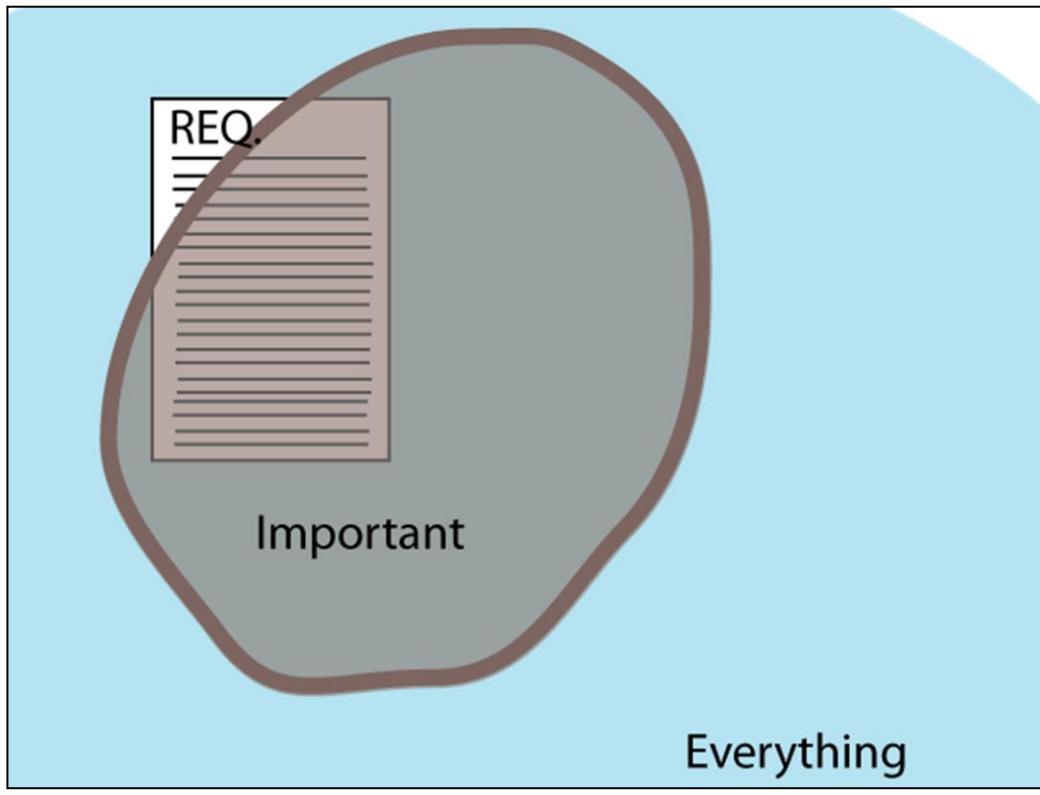
They disregard what is common, error-prone, popular, risky, changed…

They don't capture what is important



I wrote a little book about test design that have received 5.000 hits in two months.

When someone challenges authorities, you should ask: "say you're right, what can you do with this knowledge?"



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.

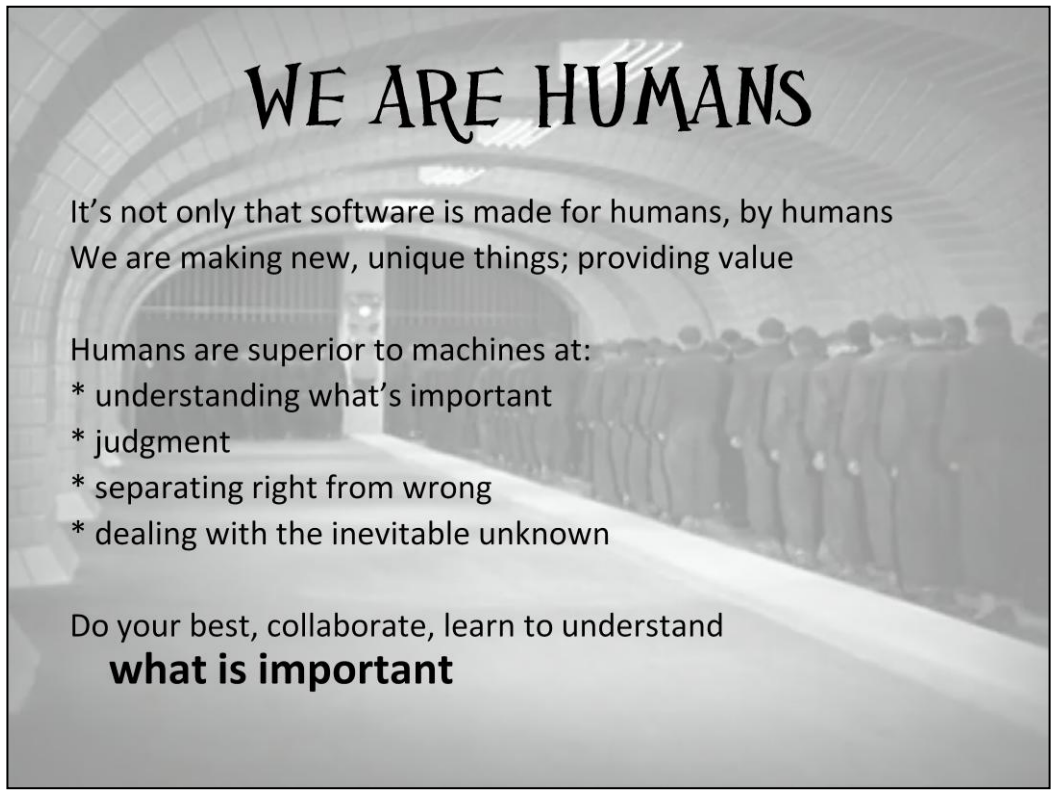
This problem has been solved many times at many places with many different approaches.
What is common could be that testers learn a lot of things from many different sources, combine things, look at many places, think critically and design tests (in advance or on-the-fly) that will cover the important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

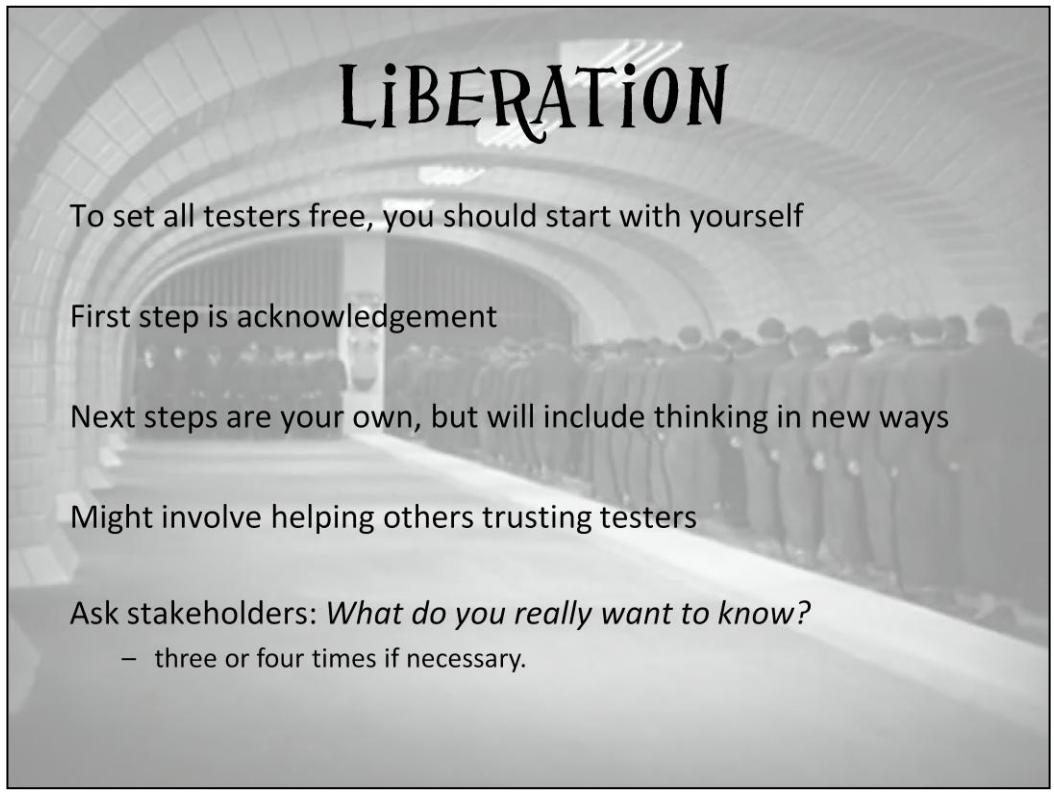
The potato is now filled with content from diverse sources.

This can be how the software world looks like for free testers.



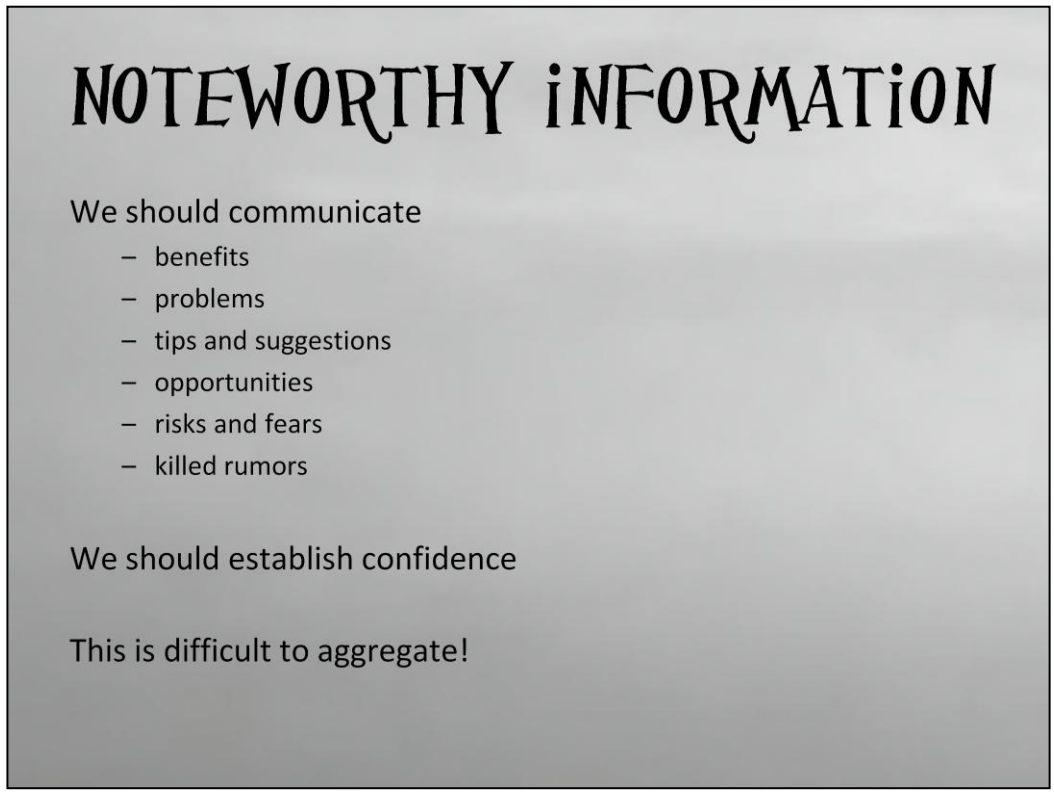
We can investigate software as humans, make subjective judgments and handle the inevitable unknown.

We don't have to look at things in binary ways, since that's not what reality is.



Liberation of our thinking.

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



The purpose of testing isn't to make it easy to create fancy reports,

it is to communicate important information.

you need knowledge about details, and the ability to summarize and explain in a variety of manners.

you need to know the software details, because then you can say better things

If communicating the essence is too difficult, maybe there are other large-scale problems, and the project should be split in more manageable pieces.

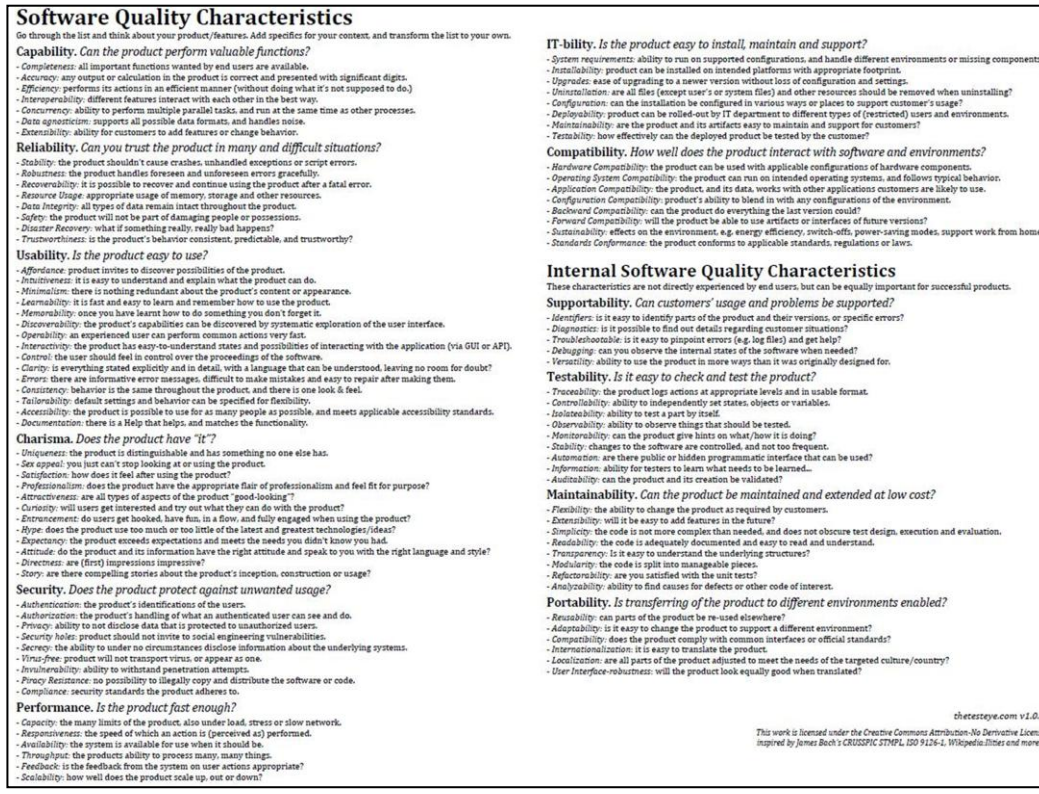
COMMUNICATION

Do we know how to communicate the essence fast?
We must train analyzing and communication (for testing!)

We need more words, and better metaphors

- saturation
- quality has many faces
- things connected to life, not machines
- your appropriate words that build confidence and trust

A shared customized quality model can help



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

Discuss with stakeholders which characteristics that are most relevent in your situation.

Let all testers have these in the back of their head.

GOING FORWARD

My steps are lighter since I cured myself

Testing isn't easy
If you make it easy, you lose the best parts

Life isn't about ticking off check boxes
It is much richer…

QUESTIONS

???

References:

- Adaptive Thinking (Gigerenzer)
- Software Testing is a Social Science (Kaner)
- The Little Black Book on Test Design (Edgren)

http://thetesteye.com/blog/

rikard.edgren@qamcom.se
QAMCOM Karlstad

Gigerenzer: Adaptive Thinking.

Kaner: Software Testing is a Social Science,
http://www.kaner.com/pdfs/KanerSocialScienceTASSQ.pdf

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