Test Strategy – Next Level

EuroSTAR Software Testing Conference | #esconfs

Rikard Edgren
rikard.edgren@learningwell.se

Test strategy – Barnum Example

- We will test the new functionality as deep as possible, and the old functionality more briefly.
- We will primarily use specifications and up-to-date risk analysis.
- As time permits, we will create automated regression tests.
- Results will be reported continuously.

- The problem with this strategy is that it is too general, and says virtually nothing.
- Your strategy needs details to be useful.

Exercise: Your test strategy

the obvious	quality characteristics	how testers think
testing mission	risks	not included
stakeholders	testers	challenges
test methods	test levels/test phases	priorities
oracles	motivations	logistics
information sources	test ideas	explanations
models	test tools	reporting

Goals for ”Next Level”

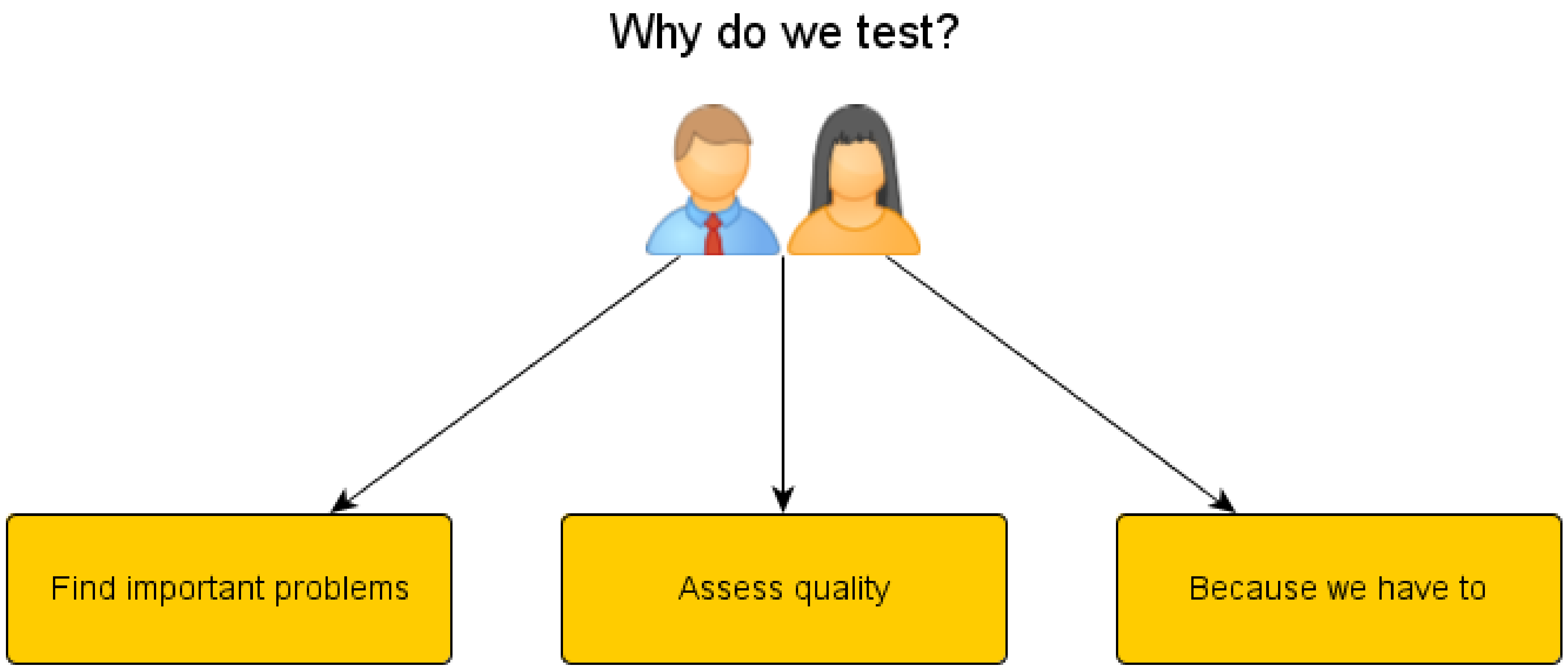
- More awareness of your implicit test strategies
- More mental tools for diversified strategies
- Ability to communicate the test strategy

Agenda

- Your test strategy decides how good your testing will be.
- But first, we need to cover how to get there:

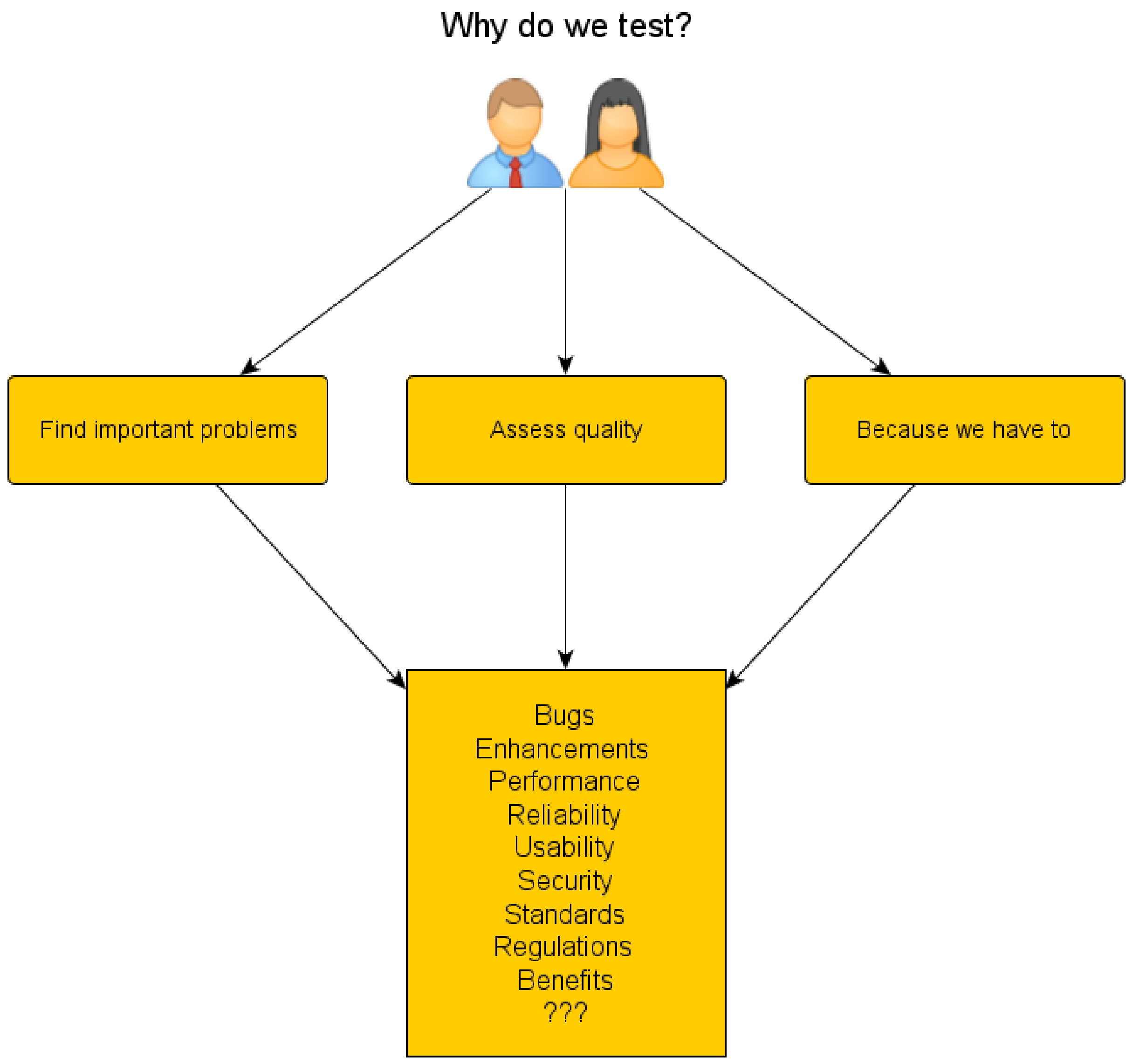
1. Testing Mission
2. Product Analysis
3. Information Sources
4. Quality Characteristics
5. Project Environment
6. Test Strategies

Different testing missions



- Different missions lead to different testing.

...similar goals



Testing Mission

- If you don't know what value testing brings, it is very difficult to do good testing (My Biggest Testing Mistake)

- Definition: Testing mission is the answer to **Why do we test?**
- The mission is given by **people**, do you know who they are?

- Really bad example:
  - The test department is responsible for testing the product

- Vague examples:
  - Contribute by finding important problems
  - Provide quality-related information (decision support)

Better Mission Example

- Documented missions for an off-the-shelf software:
  - Find problems we want to fix before release
  - Provide information for release decisions

- Un-documented testing missions:
  - Fast feedback to developers and product owners, so they can move faster
  - Identify easy, yet valuable enhancements
  - Verify that product meets promised accessibility standard
  - Inform project manager about risk status

- The importance of these missions changed over time.
- The first mission: ”find important problems” was always the most important, and it usually is.

The “so” trick

- When you have a vague mission, like
  test the product
- Add ”so” and add details:
  so we can find important problems
- Perhaps once more:
  so they can be addressed to get happier customers and fewer support calls

- Then you are closing in on a meaningful mission, where stakeholders can add more information:
  so we can take well-informed decisions,
  so product risks have been explored, so we don’t get unpleasant surprises

- Also try small additions like ”even if”, ”unless”, ”exampled by” etc.

Words requiring investigations

- **Important problems** can be elaborated with **examples**:
  - Patches
  - Complaints
  - Bad reviews
  - Embarassments
  - Bugs

- Or by **guidelines**
  - Quality objectives
  - Error catalogue
  - Checklists
  - Requirements
  - Case studies
  - Standards

- Conversations often works best to really understand.

Identify objectives & information needs

- Who are the stakeholders?
  - Project owner(s)?
  - Customers/users?
  - Project members?
  - Hidden stakeholders?
- What objectives do these stakeholders have?
  - These objectives should guide the project, and meeting them probably means a successful project.
- What information
  - are these stakeholders in need of or interested in?
  - can testing provide the project with?
  - can help us in order to meet the objectives?
- What does ”**important problems**”, ”**quality**”, ”**risks**” mean to them?

Detailed testing missions

- It can be very good with detailed testing missions:
  - Investigate if web site can handle expected load for Christmas
  - Try to find security problems for login and user accounts
  - We can't afford any more support calls regarding incorrectly filled forms; test error handling and clarity for Grandma

- But,
  - Details might obscure the whole picture and what's most important
  - What you say you want, might not be what you need

Exercise: Your stakeholders

- Who are your most important stakeholders?
  - Write their names!
  - Talk to them when you get back to work

- What do they value?
- What are they afraid of?

Product Analysis - SFDIPOT modeling

- A great framework for getting structure to your understanding of a product is to use SFDIPOT from James Bach's Heuristic Test Strategy Model.

- **Structure** – what the product is
- **Functions** – what the product does
- **Data** – what the product operates on
- **Interfaces** – how you interact with the product
- **Platform** – the environment the product depends on
- **Operations** – what the users want to accomplish
- **Time** – relations between the product and time

- These guidewords structure your thinking, and give better breadth.
- But you still have to do all the work yourself...

Product Analysis Example

- An SFDIPOT model can be thorough and time-consuming, but also fast to get an overview.
- Let’s do one together for a product of your choice.

- **Structure** – what the product is
- **Functions** – what the product does
- **Data** – what the product operates on
- **Interfaces** – how you interact with the product
- **Platform** – the environment the product depends on
- **Operations** – what the users want to accomplish
- **Time** – relations between the product and time

Many Information Sources

- The reason you should learn and use many information sources is simply that one isn’t enough.
  - Requirements only –> confirmations
  - Yourself -> opinions

- Using and choosing wisely will help design a test strategy that **have the chance** of finding important information.

- Essence of Testing: find out what's important, and test it

PRODUCT

Sources For Test Ideas

1. **Capabilities** – requirements, examples et.al.
2. **Failure Modes** – "what if…" - question everything
3. **Models** – many, if invisible models count
4. **Data** – exploit dependencies
5. **Surroundings** – environment / granularity
6. **White Box** – developer perspective + tester mindset
7. **Product History** – error catalogues
8. **Rumors** – kill them or prove them right
9. **Actual Software** – gulp your Pommac
10. **Technologies** – things that tend to go wrong
11. **Competitors** – also in-house, analogue solutions

Sources For Test Ideas

BUSINESS

12. **Purpose** – benevolent start
13. **Business Objectives** – product vision, value drivers
14. **Product Image** – what should/would users think?
15. **Business Knowledge** – learn, or co-operate
16. **Legal Aspects** – what must be avoided?

TEAM

17. **Creative Ideas** –products worth building are unique
18. **Internal Collections** – product-specific quicktests
19. **You** – you are a user, you matter

Exercise: Your Information Sources

- Write down one or two actual sources of information that improved your test strategy (do not use requirements!)
- Example:
  - I was about to do automated testing for a health care journal system. So I asked nurses that were responsible for regression testing:
  - What is risky?
  - What is boring?
  - What is difficult?

PROJECT

Sources For Test Ideas

20. **Project Background** – what happened last time?
21. **Information Objectives** – the purposes of testing
22. **Project Risks** – test risky areas early
23. **Test Artifacts** – other's testing
24. **Debt** – test against shortcuts
25. **Conversations** – people talk and collaborate
26. **Context Analysis** – what should effect your testing?
27. **Many Deliverables** – test objects and/or inspiration
28. **Tools** – a starting point for exploration

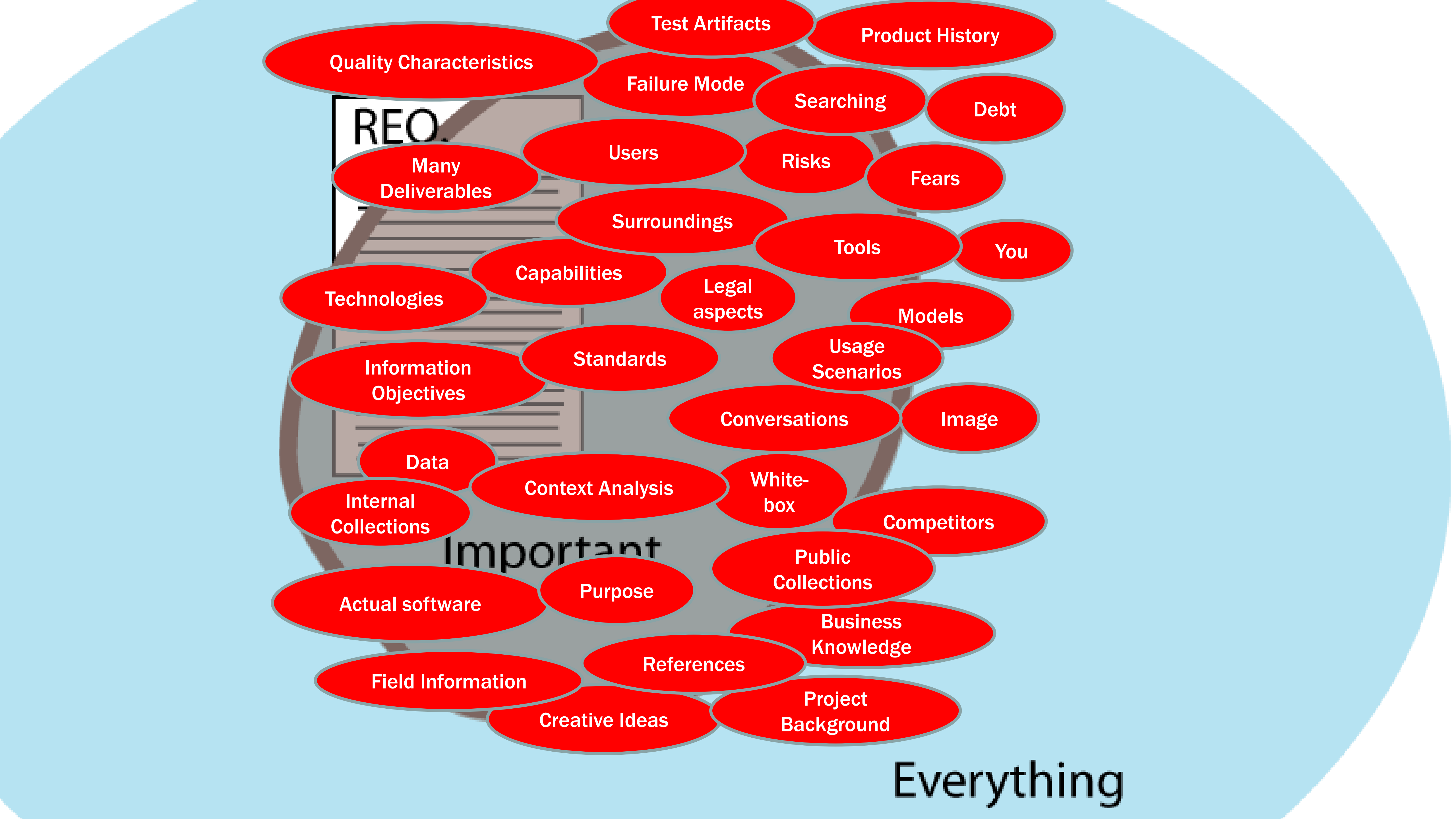
Sources For Test Ideas

STAKEHOLDERS

29. **Quality Characteristics** – in the back of your head
30. **Product Fears** – capture stakeholder's worries
31. **Usage Scenarios** – what people want to do
32. **Field Information** – environment, needs, feelings
33. **Users** – some we like, some we don't like

EXTERNAL

34. **Public Collections** – Appendix, Cheat Sheet, Not Done
35. **Standards** – read, understand, use…
36. **References** – as oracle and inspiration
37. **Searching** – Altavista, Volunia et.al.


Test Artifacts
Product History
Quality Characteristics
Failure Mode
Searching
Debt
REQ.
Users
Many Deliverables
Risks
Fears
Surroundings
Tools
You
Capabilities
Technologies
Legal aspects
Models
Usage Scenarios
Information Objectives
Standards
Conversations
Image
Data
Context Analysis
White-box
Internal Collections
Competitors
Important
Public Collections
Actual software
Purpose
Business Knowledge
References
Field Information
Creative Ideas
Project Background
Everything

Homework: Information Sources

- Go through the 37 sources, and for each one, ask yourself:
  - Should we use this one?
  - Do we already have it?
  - Should we get more information?

http://thetesteye.com/posters/TheTestEye_SourcesForTestIdeas.pdf

Test Analysis Questions

- Yes, we have all of these, but what should we do with them?
- Understand, and use as appropriate
  - Some become straightforward test ideas
  - Some need a lot of elaborations
  - Some make other tests richer

- We don’t even have time to do the requirements-based tests; how should we have time for all of these?
- Judgment, some of these give more important information
  - skip the existing tests someone (you?) already has run
  - try a few that looks promising
  - change the ways you test from time to time

Quality Characteristics

- The reason you should learn more about this is
  - to understand what's important about your software
  - to quickly generate risk-based strategie and test ideas
  - for better communication

- Definition: Quality characteristics describe desirable attributes of the system.

- Bad example: Usability is top priority
- Better example: Important customers use this software frequently, so common operations needs to be very fast.   (Operability)

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

Capability. *Can the product perform valuable functions?*

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

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

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

Usability. *Is the product easy to use?*

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

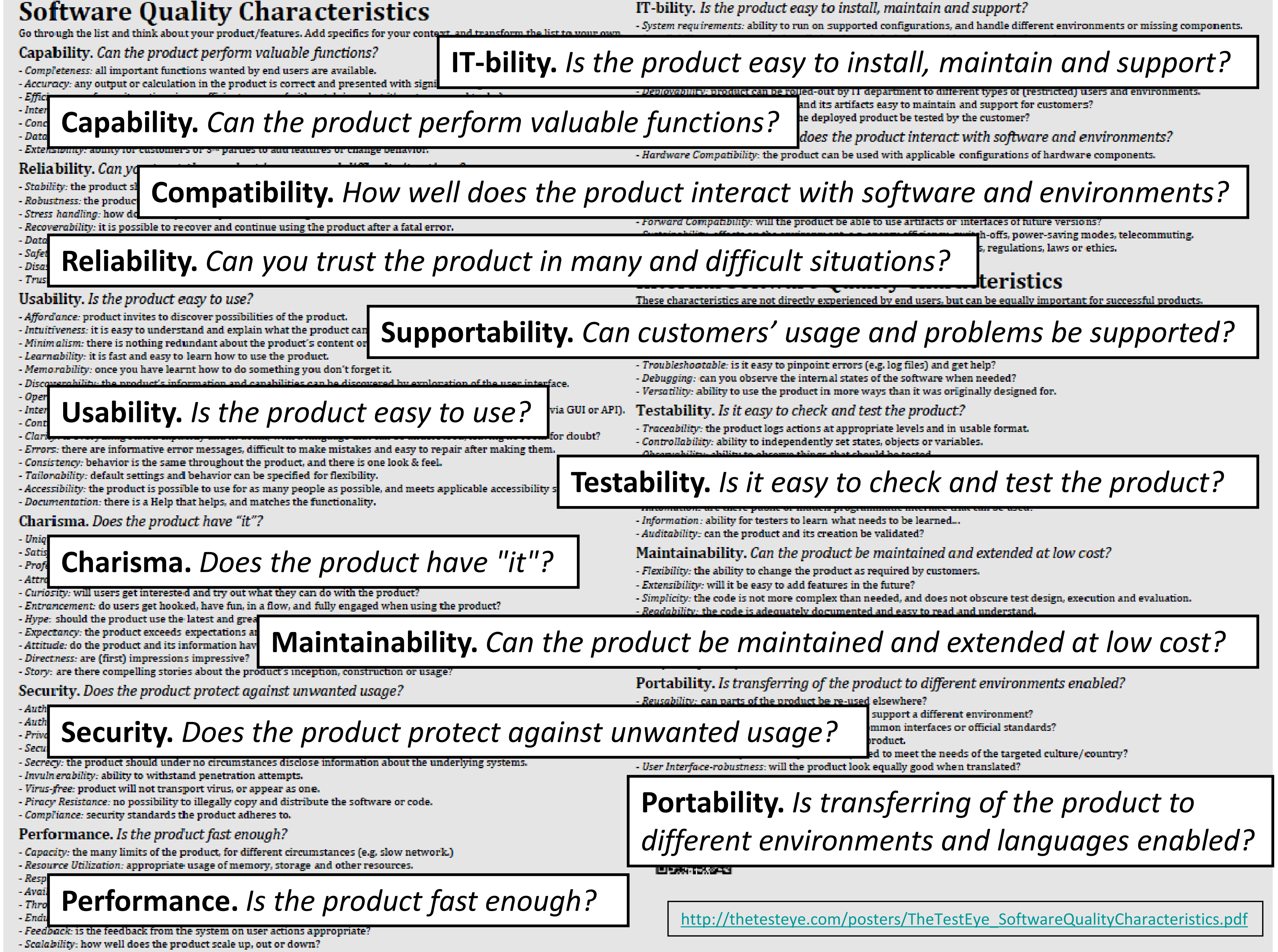
Charisma. *Does the product have "it"?*

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

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

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

Performance. *Is the product fast enough?*



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 signi
- *Effici*
- *Inter*
- *Conc*
- *Data*
- *Extensibility:* ability for customers or 3rd parties to add features or change behavior.

Reliability. *Can yo*
- *Stability:* the product sh
- *Robustness:* the produc
- *Stress handling:* how do
- *Recoverability:* it is possible to recover and continue using the product after a fatal error.
- *Data*
- *Safet*
- *Disas*
- *Trus*

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
- *Minimalism:* there is nothing redundant about the product's content or
- *Learnability:* it is fast and easy to learn how to use the product.
- *Memorability:* once you have learnt how to do something you don't forget it.
- *Discoverability:* the product's information and capabilities can be discovered by exploration of the user interface.
- *Oper*
- *Inter* via GUI or API).
- *Cont*
- *Clarity* 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 s
- *Documentation:* there is a Help that helps, and matches the functionality.

Charisma. *Does the product have "it"?*
- *Uniq*
- *Satis*
- *Profe*
- *Attr*
- *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:* should the product use the latest and grea
- *Expectancy:* the product exceeds expectations a
- *Attitude:* do the product and its information hav
- *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?*
- *Auth*
- *Auth*
- *Priv*
- *Secu*
- *Secrecy:* the product should under no circumstances disclose information about the underlying systems.
- *Invulnerability:* ability to withstand penetration attempts.
- *Virus-free:* product will not transport virus, or appear as one.
- *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, for different circumstances (e.g. slow network.)
- *Resource Utilization:* appropriate usage of memory, storage and other resources.
- *Resp*
- *Avai*
- *Thro*
- *Endu*
- *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.
- *Deployability:* product can be rolled-out by IT department to different types of (restricted) users and environments.

and its artifacts easy to maintain and support for customers?

he deployed product be tested by the customer?

does the product interact with software and environments?
- *Hardware Compatibility:* the product can be used with applicable configurations of hardware components.
- *Forward Compatibility:* will the product be able to use artifacts or interfaces of future versions?

h-offs, power-saving modes, telecommuting.

s, regulations, laws or ethics.

teristics

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

Portability. *Is transferring of the product to different environments enabled?*
- *Reusability:* can parts of the product be re-used elsewhere?

support a different environment?

mmon interfaces or official standards?

roduct.

ed to meet the needs of the targeted culture/country?
- *User Interface-robustness:* will the product look equally good when translated?

http://thetesteye.com/posters/TheTestEye_SoftwareQualityCharacteristics.pdf

Strategy examples: Reliability

- Can you trust the product in many and difficult situations?

- **Stability**: develop a semi-realistic robot that can exercise the product over weekends…
- **Data Integrity**: …with random data and built-in data integrity validation.

- **Robustness/Stress handling**: push the product’s important limits…
- **Recoverability**: …and investigate how well it recovers after (provoked) failures.

- **Safety**: perform aggressive risk-based testing to see if the ZYX might damage people under special circumstances.

Project environment

- James Bach's CIDTESTD – Project environment
  - Customers – anyone who is a client of the test project
  - Information – about the product/project that is needed for the testing
  - Developer Relations – how you get along with the programmers
  - Test Team – anyone who will perform or support testing
  - Equipment & Tools – hardware, software, or documents required to administer testing
  - Schedule – The sequence, duration, and synchronization of project events
  - Test Items – the product to be tested
  - Deliverables – the observable products of the test project

- The more you know about the project environment, the easier it is to develop efficient test strategies.

From Bach's HTSM

Test Strategy

- Test strategy contains the ideas that guide your testing effort; and deals with **what** to test, and **how** to do it.
  (Some people mean test plan or test process, which is unfortunate…)

- It is in the combination of WHAT and HOW you find the real strategy.
  - If you separate the WHAT and the HOW, it becomes general and quite useless.

- There is always a strategy, but seldom communicated

- It is not written in order to show how smart you are, it is written to communicate your ideas to (at least) two audiences:
  - Stakeholders
  - Testers

Your unique test strategy

- Every situation requires a unique test strategy.
- You always have one, even though it isn’t documented.

- A good test strategy is
  - specific – details rather than fluff
  - practical – possible to execute with “normal” turbulence
  - justified – reaches the testing missions
  - diverse – important systems needs to be tested in many different ways
  - resource efficient – uses available resources without (too much) waste
  - reviewable – possible to understand and review, so it focus on right things
  - anchored – in management, in testers
  - changeable – to be able to deal with the unevitable unknown
  - erroneous – if it isn’t “incorrect”, it is too vague, or took too long time to write

- It is better to test pretty well in many ways, than perfect in one or two. [#283, Lessons Learned in Software Testing]

Test Strategy Example

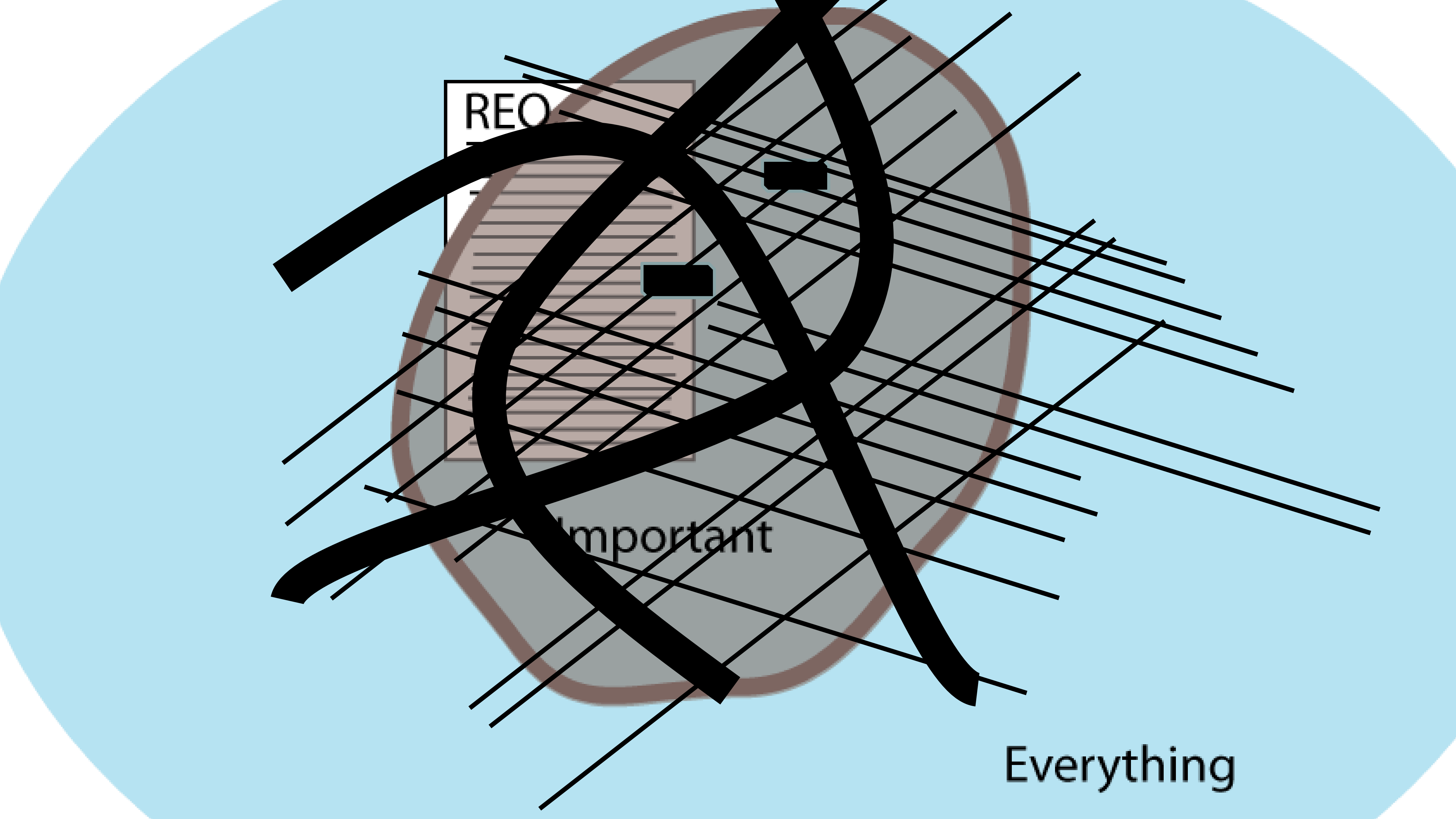
- Most important with ROPA is to help fire departments make good decisions regarding resource management. Central to this is the calculations of driving times, and accident coverage.

- We will model the product by requirements, user interface and manual, to use for basic testing of functionality.
- Since ROPA doesn’t offer support it is important to review the user documentation, and make sure error handling and other information actually helps the users.
- To test ROPA in a realistic way, we will use complex scenarios that also investigate reliability and usability.
- As a a complement, risk-based testing will be performed against secrecy, installation and data integrity (look carefully at database transactions, and visually analyze the content.)
- As the product hasn’t previously been tested by ”testing professionals”, a list of bugs is an important deliverable (there exists a list of 10 known issues that we will investigate at once.)
- To facilitate future testing, the testers should give guidelines for testability improvements, e.g. programmatic interfaces that allow automatic regression testing of calculations.

- Challenge: Currently we have no really good oracle (except sanity and Google Maps) to decide whether the driving times are accurate.

Example of test strategy activities

- These ”test strategies” were used at a product company:
  - Unit testing: 75% code coverage for new code
  - Automated regression testing of API
  - Automated regression testing of selected bugs
  - Manual Smoke Pass (once a month)
  - Automated Smoke Pass (every day)
  - Detailed test cases, at least one for each requirement
  - Vague test cases, at least one for each requirement
  - Session-based exploratory testing (on chosen risks)
  - Totally free testing (Brolin-role)
  - Scenario testing with several people, scenario created on-the-fly
  - Security testing (without being penetration experts)
  - Performance testing with inhouse framework
  - Investigation of interesting/important support incidents
  - Usability testing with students
  - User testing with focus group (real users)
  - Acceptance testing by product owner
  - A lot of installation/upgrade testing…
  - Verification of fixed bugs, and testing for side-effects
  - Code review of sensitive parts
  - Test code right after it has been written


REO
Important
Everything

Aspects of test strategies

- the obvious
- testing mission
- stakeholders
- test methods
- oracles
- information sources
- models
- quality characteristics
- risks
- testers
- test levels/test phases
- motivations
- test ideas
- test tools
- how testers think
- not included
- challenges
- priorities
- logistics
- explanations
- reporting

Product and project risks

- Product risks
  - Found everywhere in today’s material
  - Especially in Quality Characteristics
  - Has a 90’s feeling to me, but there is nothing wrong with a risk-centered strategy

- Project risks
  - Why won’t your strategy work?
  - Found in details
  - Found in Project Environment

- As with everything else, it is in the details and your understanding…

General testing techniques

- **Function testing** – test that each function does with it's supposed to
- **Risk-based testing** – try to provoke important risks (deal with probablility afterwards)
- **Specification-based testing** – use product claims (not necessarily a specification) and see if they hold.
- **Scenario testing** – test longer sequences, with complexity for sequence order, users, data and/or environment.
- **Model-based testning** – test from states, architecture, flows or custom models.
- **Quality objective-based testing** – Each quality characteristic can be used as a testing method, e.g. performance, security, usability, compatibility (plus sub-categories.)
- **High volume testing** – Run an awful amount of tests to evaluate stability, use of "all" data, see patterns etc.
- **Domain testing** – Choose data from equivalence groups, boundary values, or best representatives.
- **User testing** – Let (simulated) users perform tasks.
- **Testing without flourishes**– You know what to test, and do it.

- Manual/Automated/Exploratory/Scripted are orthogonal.

Exercise: FizzBuzz Test Strategy

- This program is an exercise for software testers.
  - http://www.thetesteye.com/code/FizzBuzz.rb
  - http://www.thetesteye.com/code/FizzBuzz.exe.zip (Windows only)
- As input it takes an integer between 1 and 1000, and repeats it as output.
- If the number is a multiple of three, it should print "Fizz" instead of the number and for the multiples of five print "Buzz".
- For numbers which are multiples of both three and five it should give "FizzBuzz" as output.

- Your testing mission is to find any threats to this software being a useful testing exercise for testers around the world.

- What would be a good test strategy?

FizzBuzz Test Strategy

I want to perform the testing I think testers will do.

a. I would start by executing and getting a feel of it. Usability aspects will be evaluated, as well as noting interesting behavior.
b. I would do manual samples of fizz, buzz, fizzbuzz, number, too high, negative, way too high, too much input, strings, special words (fizz, ruby, null)
c. I would proof-read all text, including log file
d. Pay a lot of attention to testability, especially test the content of log file
e. I would review the code
f. I would get a handful of testers to do the exercise to see how useful, and inspiring it is
g. Hopefully these testers have diverse platforms, but some additional operating systems and Ruby versions should also be tested.
h. I would write my own program that produces the same output, to check that all 1000 values are correct. Tests correctness, stability, endurance, and is a bit of fun as well. Feed these values into unit tests. (I have two examples of this; one with AutoHotkey, and one with Ruby unit tests.)
i. I would run many inputs with AutoHotkey, both valid and invalid, to see endurance and robustness.
j. I would try to talk to someone knowledgable to make sure the requirements are good, and correctly understood by me.

Tying things together

- There are many things that are important, and many ways to test them.

- Some testing activities will cover many important aspects.
- Some important aspects require several testing activities.

- You don’t know the details of the HOWs, but you can communicate them at an appropriate level.

- You might also include WHY, also for marketing purposes.

Anchored in...

- Situation
  - Test what is demanded by the context.
- Management
  - Test to get the information others need.
- Testers
  - Make sure testers know where you are aiming, and why.
- At the same time adjustable, since things always change...

Always with a flavor of...

- ...risk judgment
  - So you focus on what's most important
- ...test design
  - Continuously jot down fruitful test ideas
- ...communication
  - So stakeholders get the information they need
  - So testing can be improved

- *Testing is never better than the communication of the results*

Exercise: Specific test strategy

- Team up.

- Choose one of your stakeholders from previous exercise.
- Design a test strategy that will generate the information that **this specific person** needs.

- (Yes, this is not how we do it in reality, but you should practice this, it’s about focusing on information objectives.)

Homework: Diversified test strategy

- Team up.

- Come up with **plenty** of different ways to test your product.
- Suspend judgment until you run out of ideas.

Test Strategy Bias

- Answering an easier question
  - Dodging the most important questions
- What you see is all there is (WYSIATI)
  - What are others doing?
- Halo effect
  - Don’t judge by single observations
- Illusion of validity
  - Does one good example justify a test method?
- Optimistic bias
  - Downhills, sun and wind in the back?
- Focusing illusion
  - It gets more important when you think about it

- You can’t avoid bias, but you can manage it.

Inspired by Kahneman

Test Strategy QA

- Review & Conversations
- Re-visit Quality Characteristics and stakeholder needs
- Does the strategy cover what you actually do/want to do?
- Ask yourself (honestly):
  - *What will be praised?*
  - *What would the worst critic say?*

Results

- When you have developed an anchored test strategy, you have learned a lot.
- You have many ideas about what to test, and how.
- You have a starting point for reporting.

- You have stakeholders agreeing what you are up to.

If you think you have a reporting problem,
I suspect it's really about test strategy
communication.

Exercise: 30 seconds

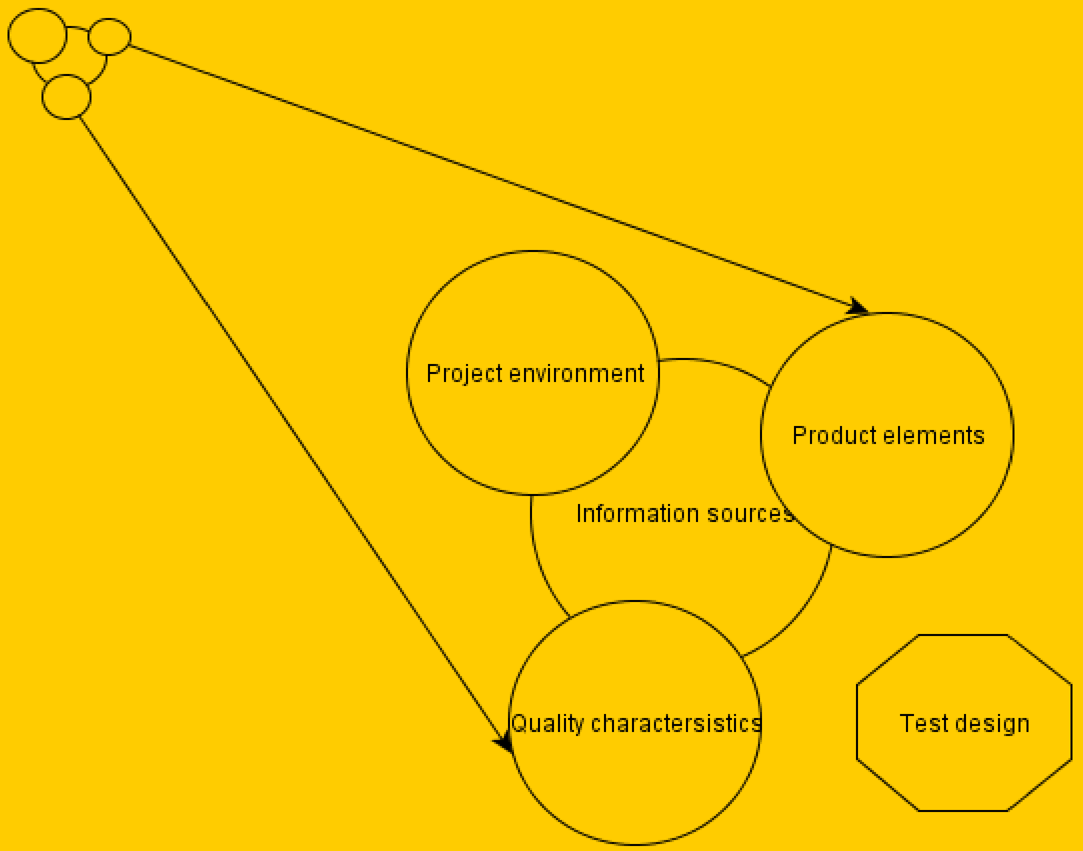
- Team up in pairs.
- Explain your current, real-world strategy in 30 seconds.
- Don’t talk too fast, focus on the most important and challenging parts.

Exercise: 90 seconds

- Team up in pairs.
- Explain your current, real-world strategy in 90 seconds.
- Don’t talk too fast, focus on the most important and challenging parts.

Testing mission

Context Analysis



grounded testing missions
diversified test strategy
test ideas
start for reporting

Summary test strategy

- Test strategy is hard; but you will use it all the time when you test.
  - *If it is easy, you probably know too little*

- The more you learn, the better your test strategy will be.

- You will get a good start if you find out about your testing, mission, project environment, product elements, information sources and quality characteristics.

- The first test strategy in the project is far from perfect.
- That is why you should modify and change your strategy whenever you learn more, and when the context changes.

Questions

- ???

- Further reading:
  - Bach: Heuristic Test Strategy Model
    http://www.testingeducation.org/BBST/foundations/Bach_satisfice-tsm-4p-1.pdf
  - Kaner, Bach, Pettichord: Lessons Learned in Software Testing
  - Edgren: The Little Black Book on Test Design
    http://www.thetesteye.com/papers/TheLittleBlackBookOnTestDesign.pdf
  - Edgren: Den lilla svarta om teststrategi (in Swedish)
    http://www.thetesteye.com/papers/DenLillaSvartaOmTeststrategi.pdf

DEN LILLA SVARTA OM TESTSTRATEGI

RIKARD EDGREN & HENRIK EMILSSON

www.thetesteye.com

rikard.edgren@learningwell.se