

# Introduction to Test Strategy

Scandinavian Developer Conference, 5-mar-2013

Rikard Edgren  
Qamcom Karlstad  
rikard.edgren@qamcom.se

Why?



How?

# Agenda

## 1. Testing Mission

The reasons for testing.

## 2. Context Analysis

Finding out what's important.

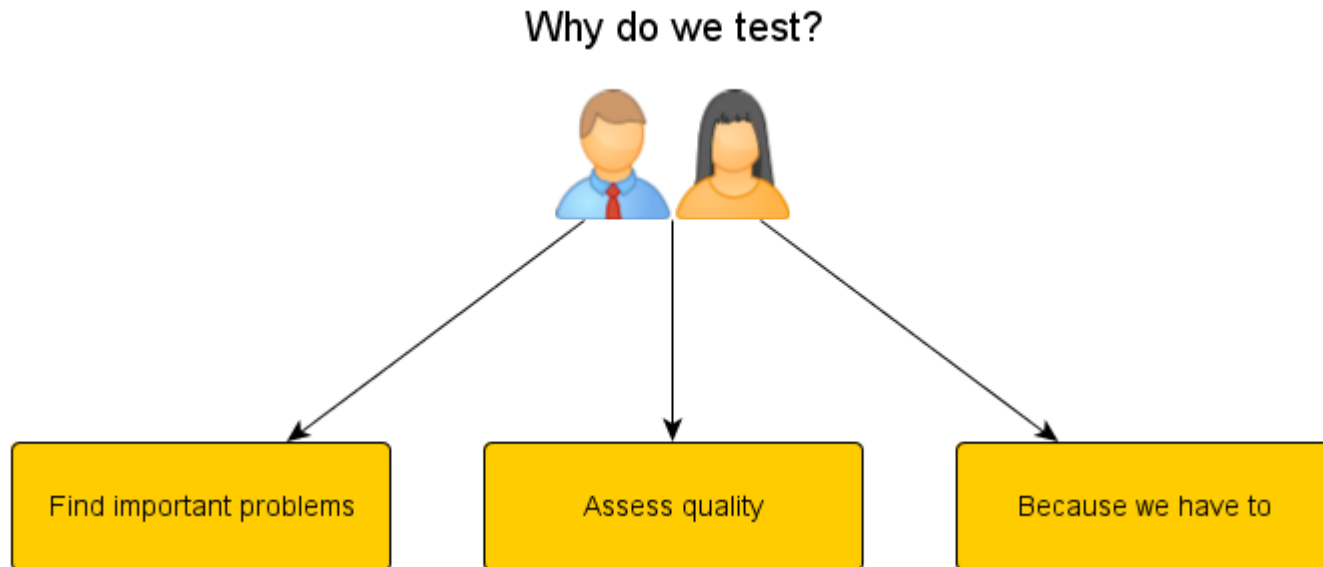
## 3. Test Strategy

What to test, and how.

# 1. Testing Mission

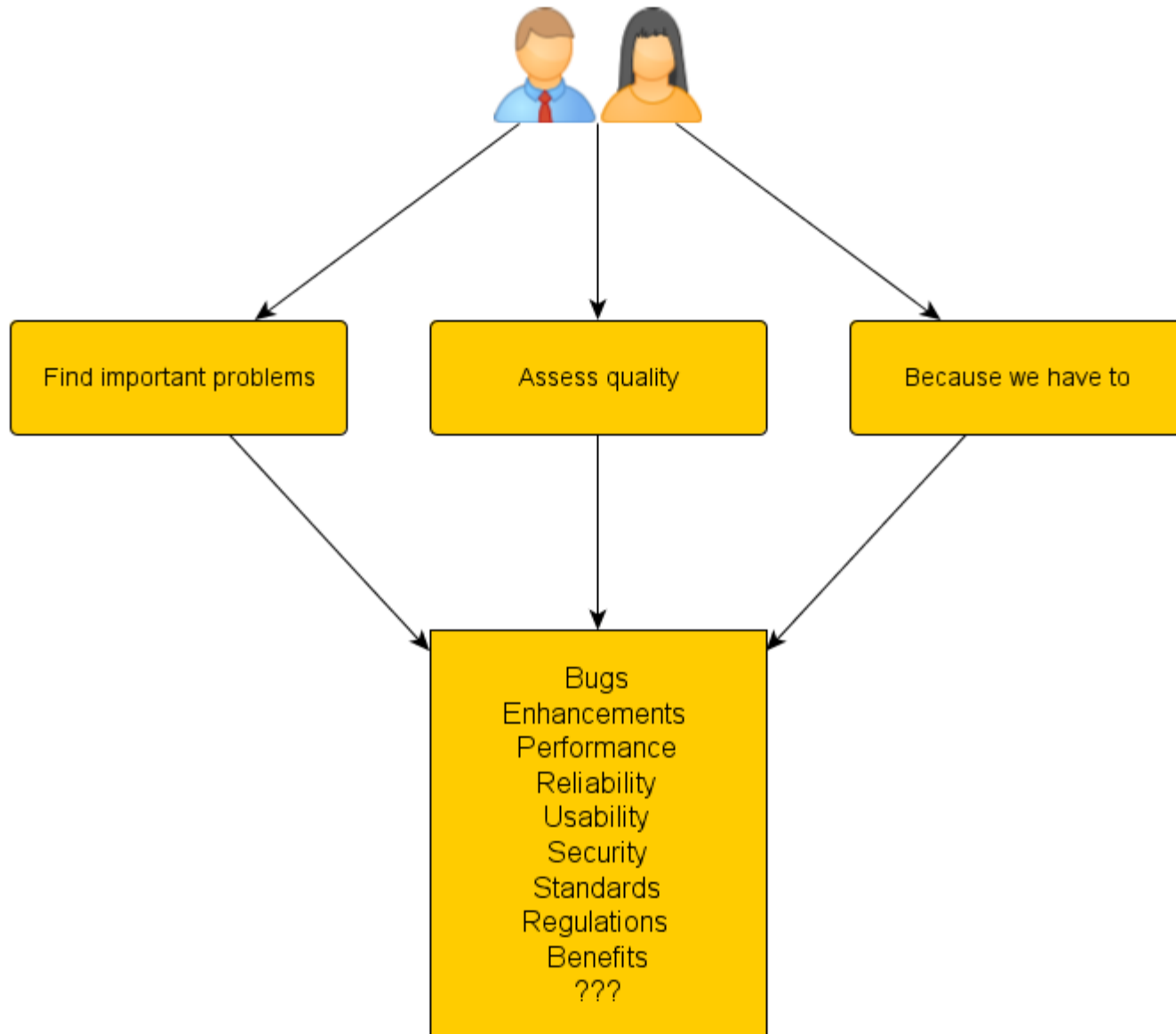
- ▶ Purpose to learn: If you don't know what value testing brings, it is very difficult to do good testing.
- ▶ Definition: Testing mission is the answer to **Why do we test?**
- ▶ The mission is given by **people**, do you know who they are?
- ▶ Decent examples:
  - Contribute by finding important problems
  - Provide quality-related information (decision support)
- ▶ Bad example:
  - The test department is responsible for testing the product

# Different testing missions



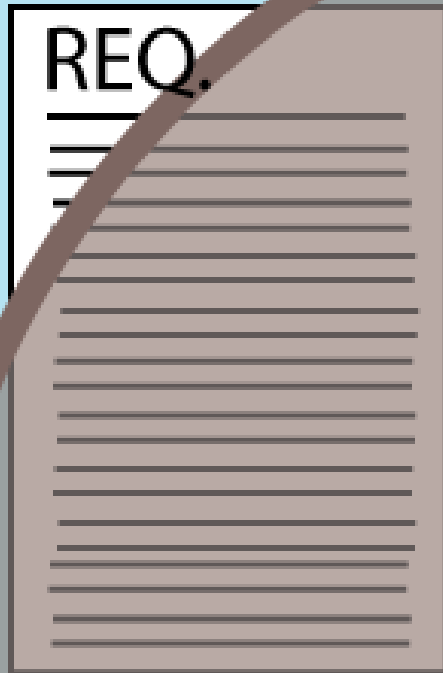
- Different missions lead to different testing.

## Why do we test?



# 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*
- ▶ ...and of course the testing will be better if we know what the information will be used for!



Important

Everything



# Find out what's important

- ▶ Talk to stakeholders.
- ▶ Ask "What do you want to know?", many times if necessary.
- ▶ Investigate relevant information sources:
  - Specifications
  - Quality objectives
  - Fears
  - Technologies
  - Business knowledge
  - Real customers
  - ..., see [37 Sources for Test Ideas](#)

# Words requiring investigations

## ► **Important problems** can be elaborated with **examples**:

- Patches
- Complaints
- Bad reviews
- Embarrassments
- Bugs

## ► Or by **guidelines**

- Quality objectives
- Error catalogue
- Checklists
- Requirements
- Case studies
- Standards

## 2. Context Analysis

- ▶ How should the project environment effect the testing?
  - Other's testing
  - Developer/Tester interaction
  - Test environment
  - Planning and deliverables
- ▶ What should be tested?
  - Create models (James Bach's SFDIPOT is very powerful)
- ▶ Which quality characteristics matter?

Based on James Bach's HTSM

# 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 figures.
- **Efficiency:** the product uses resources in an optimal way.
- **Interoperability:** the product can be used with other products or systems.
- **Compatibility:** the product can be used with applicable configurations of hardware components.
- **Extensibility:** ability for customers or 3<sup>rd</sup> parties to add features or change behavior.

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

- **Stability:** the product does not crash or stop working.
- **Robustness:** the product can handle unexpected inputs or situations.
- **Stress handling:** how does the product behave under heavy load or stress?
- **Recoverability:** it is possible to recover and continue using the product after a fatal error.
- **Data integrity:** the product ensures that data is not lost or corrupted.
- **Safety:** the product does not cause harm or damage.
- **Disaster recovery:** the product can be restored after a disaster.
- **Trust:** the product is reliable and secure.

## 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 interface.
- **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.
- **Operability:** the product can be used via GUI or API.
- **Interactivity:** the product responds to user actions.
- **Content:** the product has useful and relevant content.
- **Clarity:** the product uses clear and simple language.
- **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 different from other products.
- **Satisfaction:** the product meets user expectations.
- **Professionality:** the product looks and feels professional.
- **Attraction:** the product is visually appealing.
- **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 greatest technology?
- **Expectancy:** the product exceeds expectations and delivers on promises.
- **Attitude:** do the product and its information have a positive attitude?
- **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 ensures that users are who they claim to be.
- **Authorization:** the product ensures that users can only access the resources they are allowed to.
- **Privacy:** the product ensures that user data is protected and not shared with third parties.
- **Security:** the product is protected against unauthorized access and data loss.
- **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.
- **Responsiveness:** the product responds quickly to user actions.
- **Availability:** the product is available when needed.
- **Throughput:** the product can handle a large number of requests.
- **End-user satisfaction:** the product meets user expectations.
- **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.

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

- **Removability:** product can be rolled-out by IT department to different types of (restricted) users and environments.
- **Supportability:** can customers' usage and problems be supported?
- **Testability:** is it easy to check and test the product?
- **Maintainability:** can the product be maintained and extended at low cost?
- **Portability:** is transferring of the product to different environments enabled?
- **Security:** does the product protect against unwanted usage?
- **Performance:** is the product fast enough?
- **Charisma:** does the product have "it"?
- **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?
- **Interoperability:** will the product be able to interact with other products or systems?
- **Compatibility:** will the product be able to interact with software and environments?
- **Extensibility:** will the product be able to be extended with new features or functionality?
- **Scalability:** will the product be able to scale up, out or down?
- **Reliability:** will the product be able to handle different environments or missing components?
- **Stability:** will the product be able to handle different environments or missing components?
- **Robustness:** will the product be able to handle different environments or missing components?
- **Stress handling:** will the product be able to handle different environments or missing components?
- **Recoverability:** will the product be able to handle different environments or missing components?
- **Data integrity:** will the product be able to handle different environments or missing components?
- **Safety:** will the product be able to handle different environments or missing components?
- **Disaster recovery:** will the product be able to handle different environments or missing components?
- **Trust:** will the product be able to handle different environments or missing components?

## 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 enabled?

- **Reusability:** can parts of the product be re-used elsewhere?
- **Interoperability:** will the product be able to interact with other products or systems?
- **Compatibility:** will the product be able to interact with software and environments?
- **Extensibility:** will the product be able to be extended with new features or functionality?
- **Scalability:** will the product be able to scale up, out or down?
- **Reliability:** will the product be able to handle different environments or missing components?
- **Stability:** will the product be able to handle different environments or missing components?
- **Robustness:** will the product be able to handle different environments or missing components?
- **Stress handling:** will the product be able to handle different environments or missing components?
- **Recoverability:** will the product be able to handle different environments or missing components?
- **Data integrity:** will the product be able to handle different environments or missing components?
- **Safety:** will the product be able to handle different environments or missing components?
- **Disaster recovery:** will the product be able to handle different environments or missing components?
- **Trust:** will the product be able to handle different environments or missing components?
- **User Interface-robustness:** will the product look equally good when translated?

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

## Performance. Is the product fast enough?

# 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.

# 3. Test Strategy

- ▶ Purpose: The strategy drives the testing, in order to reach the testing mission.
- ▶ Definition: Test strategy consists of guidelines and ideas that describe **what** should be tested, and **how**.
- ▶ (Some people mean test plan or test process.)
- ▶ It is not written in order to show how smart you are, it is written to communicate to (at least) two audiences:
  - Stakeholders
  - Testers

# 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-do-date risk analyses.
  - ▶ 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.

# 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 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.



# Your unique test strategy

- ▶ Every situation requires a unique test strategy.
- ▶ A good test strategy is specific, justified and realistic.
- ▶ It is better to test pretty well in many ways, than perfect in one or two. [ #283, Lessons Learned in Software Testing ]

# Aspects of test strategies

- ▶ What is important?
- ▶ Goals
- ▶ Test techniques
- ▶ Test ideas (worth mentioning)
- ▶ Information sources
- ▶ Oracles
- ▶ Models
- ▶ Quality objectives
- ▶ How testers think
- ▶ Trade-offs
- ▶ Risks

# 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...

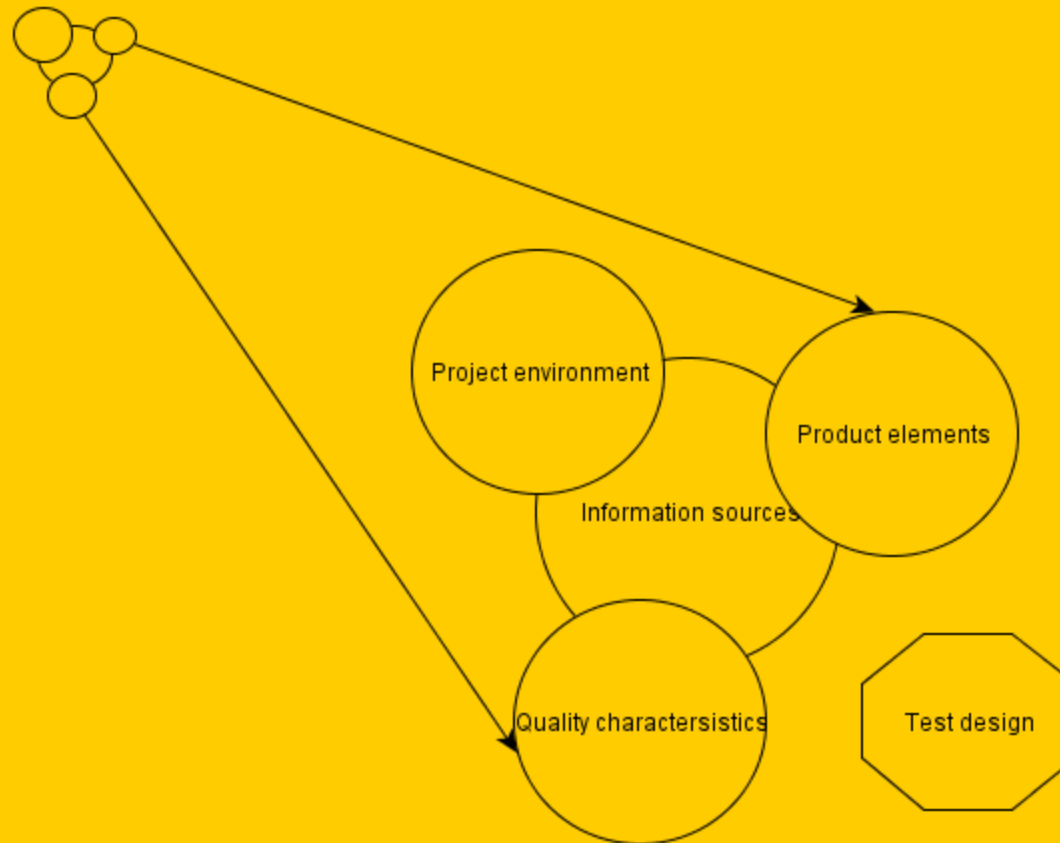
# Always with a flavor of...

- ▶ ...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*

# 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.

# Context Analysis



grounded testing missions  
diversified test strategy  
test ideas  
start for reporting

# 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.

# Finale

- ▶ It's about the information you gather, and share.
- ▶ It's about how you think.
- ▶ You need to find your test strategies for **your context**.
- ▶ Do your best, collaborate, learn to understand **what is important**.



# Questions

▶ ???

▶ Literature:

- [Heuristic Test Strategy Model](#) (James Bach)
- [BBST Test Design](#) (Kaner, Fiedler)
- [The Little Black Book on Test Design](#) (Edgren)



[www.thetesteye.com](http://www.thetesteye.com)

[rikard.edgren@qamcom.se](mailto:rikard.edgren@qamcom.se)