Surviving in a QA-Organisation

Markus Clermont
Google Ireland Ltd.

November 17, 2006
QA is a bandaid, all Google engineers are quality engineers

Alan Warren,
Google Director of Engineering
What does that imply?

- QA is recognized to be of major concern.
- There exist a wide understanding that testing is important.

*BUT:*

- ”We work in an extremely fast paced environment”.
  - Short release cycles result in incomplete or out-of-date documentation.
  - Many features are added in the last minute.
  - Attitude towards documentation.
  - Sometimes we simply run out of time...

- ”We work in a flexible environment”.
  - Following a structured development process is perceived to be inefficient and actually harmful.
  - There is not many artefacts we can rely on being actually there.
How can we handle that?

- Exploit the short feedback loops.
- Develop clever tools for testing
  - For each project, determine how far automation should be driven.
  - Determine the minimal amount/formality of documentation necessary.
- Many interesting questions lie on the way that have to be answered:
  1. What are the goals of testing?
  2. Which approach(es) shall we apply?
  3. How much testing do we have to do to achieve our goals, and how much can we do?
    - Considering time- and cost restrictions, how can we prioritize test cases?
  4. How can we communicate the benefits to project-management and developers?
The Challenge- and my dark little secret!

For many questions answers do exist. Usually they are not applicable, because:

- They restrict the flexibility of developers and do not consider the diversity of projects that are handled in parallel.
- The overhead simply does not pay-off, or is infeasible in first place.
- We- the test community- fail to properly communicate the benefits to our customers- the management and the developers.

Many problems are considered solved, although the solutions apply only to limited domains.

- The usual definition of testing is *to compare the behavior of a software artefact to the specified behavior.*
- I have **NEVER** in my life tested software before.
- How many of you have?
The Challenge, cntd.

- Step outside of the limited domain of well-specified, small systems. Please.
- Design flexible testing-tools that do not depend on a specific testing approach.
- Currently available process-models are too rigid to find broad acceptance. People evade them by becoming agile.
- Testing in the agile-environment
  - Dealing with the absence of documentation.
  - Short release cycles, again.
  - Little time for testing.
  - Developers usually working on the next release, when the current one is under test.
Functional Testing of web-based systems adds additional issues to look out for.

1. Does the application-server transmit the correct data?
2. Does the web-server pass the data correctly on to the client?
3. Is the browser rendering the transmitted HTML correctly?
4. Embedded Java-Script.

Usually, the first and the third issue are given special attention. Case 2 is hardly ever considered. Java-Script is an issue on its own.
I do not want to give up yet

- Let us do agile testing in the agile environment.
  - Intense communication with Developers.
  - Do not rely on documents. They are never good enough.
  - Assign testers to developers, not to features.
  - Test-Doclets instead of test-plans
  - Communicate test results via dashboards. Publicly.
  - Make use of the vicinity of the customers. Develop test cases with them.

- Tests should be fully automated.
  - There exist appropriate tools for Unit-Testing.
  - We *CAN* go down the same road for functional and load testing.
    - Model-Based Testing seems to offer many feasible answers. And pitfalls.
  - How can we automate other time consuming tasks, e.g. UAT?
Testing process - Set up

- One senior QA expert to act as an advisor to the project leadership.
- One QA expert usually cooperates with a couple of developers and test exclusively the features they are working on.
- Testcases and Testoracle will be developed in parallel and in cooperation with the developers and customers.
- There is a separate QA-Build of the project that is then tested. This build is made available as soon as possible.
- Early feedback from the tester, frequent discussions with ‘his’ developers.
- High degree of automation to keep tester’s capacities free for none-routine tasks.
Testing process - IO

- **Input:**
  - Some informal requirements description,
  - Clarifications with Developers and Customers,
  - Domain-Knowledge
  - Aims.
  - Time- and resource restrictions.

- **Output:**
  - Acceptance criteria.
  - Testcases.
  - Scripts/ Tools for automatically testing the application.
  - Bug reports.
  - Release notes.
Applied Techniques/Tools

• Model-Based Testing
• Boundary Value Testing
  • Coverage criteria can be applied
  • Extended, domain-specific neighborhood notion
• Tools for
  • Administration of Test-Cases
  • Keeping track of bugs
  • Automatic execution of tests
    • Using a specific web-browser
    • Without any web-browser
    • Performance/Load Testing
• Keeping track of overall test-results and testing progress.
The Model

- Use-Case based approach.
- We see the application as a set of dialogs.
- Each dialog has different entry- and exit-paths.
- We figure out, which sets of inputs will yield to the selection of a certain exit-path.
- A path can be triggered by different classes of inputs (i.e. partitions of the input domain).
  Example: Checking username and password.
    - The error path will be activated for non-existant usernames, with any password, and
    - for valid usernames, with the wrong password (e.g. username/password).
- The pathes through the model (use cases) are weighted with an importance-grade.
Testdata generation and evaluation

For each input-class we generate:

- Random test data, well within the expected boundaries.
- Invalid inputs (i.e. inputs that should make me leave the path)
- Typical inputs (not generated but supplied)
- Boundary test cases
  - Given a sufficiently precise model, they are easy to generate
  - What is a boundary-value?
    An input with a neighbor outside the input-class.
  - What is a neighbor?
  - We can specify coverage criteria for boundary value testing.
Boundary Coverage Criteria

How well is each partition covered by test data?

1. One Boundary
2. All Edges
3. Multi Dimensional
4. AE / MD
5. All Boundaries

Domain: $\mathbb{Z} \times \mathbb{Z}$
Constraints: $x \geq 0 \land y \geq 0 \land x \leq 5 \land y \leq 4 \land y \leq 7 - x$
Boundary Coverage Criteria

How well is each partition covered by test data?

1. One Boundary
2. All Edges
3. Multi Dimensional
4. AE / MD
5. All Boundaries

Domain: $\mathbb{Z} \times \mathbb{Z}$
Constraints: $x \geq 0 \land y \geq 0 \land x \leq 5 \land y \leq 4 \land y \leq 7 - x$
Boundary Coverage Criteria

How well is each partition covered by test data?

1. One Boundary
2. All Edges
3. Multi Dimensional
4. AE / MD
5. All Boundaries

Domain: $\mathbb{Z} \times \mathbb{Z}$
Constraints: $x \geq 0 \land y \geq 0 \land x \leq 5 \land y \leq 4 \land y \leq 7 - x$
Boundary Coverage Criteria

How well is each partition covered by test data?

1. One Boundary
2. All Edges
3. Multi Dimensional
4. AE / MD
5. All Boundaries

Domain: $\mathbb{Z} \times \mathbb{Z}$

Constraints: $x \geq 0 \land y \geq 0 \land x \leq 5 \land y \leq 4 \land y \leq 7 - x$
Boundary Coverage Criteria

How well is each partition covered by test data?

1. One Boundary
2. All Edges
3. Multi Dimensional
4. AE / MD
5. All Boundaries

Domain: $\mathbb{Z} \times \mathbb{Z}$
Constraints: $x \geq 0 \land y \geq 0 \land x \leq 5 \land y \leq 4 \land y \leq 7 - x$
Boundary Coverage Criteria

How well is each partition covered by test data?

1. One Boundary
2. All Edges
3. Multi Dimensional
4. AE / MD
5. All Boundaries

Domain: \( \mathbb{Z} \times \mathbb{Z} \)
Constraints: \( x \geq 0 \land y \geq 0 \land x \leq 5 \land y \leq 4 \land y \leq 7 - x \)
Testdata evaluation

We can evaluate our test-data in multiple ways:

- Which dialogs did we visit?
- Did we leave dialogs on all exit-pathes?
- Did we enter dialogs via all entry pathes?
- For each entry/exit path, did we cover all input-classes?
- For each input-class, what boundary coverage do we achieve?

Just add probabilities to the pathes/input-classes, and

- the model becomes a Markov chain.
- We can calculate the reliability of the application
Architecture of a testing toolkit

Introduction

The Challenge

A possible solution?

The End...

User/QA-Engineer

Testdata Generator

Testcase-Repository

Instruction Provider

Test-Executor

Test-Executor Handler

Log

GUI

Autom.

Webbrowser

Webserver

SUT

Result-Dashboard

Testoracle

Repository

Testcase-Generator

Testdata

QA-Engineer

User/

Test Driver

Test Driver
Benefits of the Toolkit

- End-to-End Test Execution is widely automated.
- Functional tests can be initiated after any build. Only UAT requires further manual intervention.
- Test-Oracle and Models are derived in an agile fashion, on a per feature basis.
- Provides support for load and performance testing
  - Multiple Instances of the Test-Executor can be run,
  - Distributed over many machines.
  - They can follow the same or different InstructionProviders,
  - Synchronisation mechanism is provided.
- Tight collaboration between Test-Engineer and Developer.
- Independent of any particular testing method.
Of course, there are Problems, too

- Currently support only for web-based systems.
- The Testoracle has to be written manually.
  - Predicates over the html-output
  - Not too difficult, but
  - requires Java or C++ knowledge.
  - Could be replaced on the long run by a high-level language.
  - JavaScript is usually hard to handle this way.
- UAT requires, of course, user-interaction
Conclusion

- We need a mix of methods to be successful.
- Process oriented approaches are not universally applicable.
  - We have to do in Rome, as the Romans do...
- Document-driven development is fine. Just, we don’t do it.
- Good tool-support is a crucial issue.
- Some research is just far away from us, and
- A lot of research is focusing on specific problems in controlled environments. That is legitimate. It doesn’t help us, though.
- To have a QA-Organization or a QA-Process is dangerous.
The imperatives of successful Testing and QA at least, in my environment.

1. Define your goals.
2. Be a fully integrated part of the organisation.
3. Mentor developers to recognize QA issues.
4. Be flexible and accommodate to your environment.
5. Test as soon as you can.
6. Automate your tests.
7. Reuse your tests.
8. Don’t rely on documentation.
9. Don’t rely on a single approach.
10. Communicate test plans, test cases and test results.
Thank you!

Got curious? Want to contribute?

http://www.google.ie/jobs
markusclermont@google.com