1. **Introduction**

The goal of testing is to produce quality-related information for someone who needs it for something. Some make business decisions based on it; some improve the product based on it.

Metrics are an important part of that, but sometimes it is not clear what metrics to use. Model-based testing is one example of that.

This short paper describes some of the issues related to metrics, reporting and communication of product quality and testing metrics when model-based testing is used.
2. **Different tests have different purpose**

Not all testing is equivalent. Each type testing should have a defined purpose.

One view that has gained plenty of support lately and requires mentioning here is the "agile testing quadrants" by Brian Marick and thoroughly documented by Crispin & Gregory (2009).

![Agile Testing Quadrants](image)

*Figure 1. The (agile) testing quadrants by Brian Marick, described in Crispin & Gregory (2009).*
3. **Information flow**

![Diagram showing information flow]

*Figure 2. Information flow from the test system to using the information.*

4. **Goals for quality information**

General information:

- Meets the receiver’s needs.
- Supports goals.
- Different information may be needed for different parties – support for team’s work is different than support for business.
- Needs to be reliable – or deficiencies know.
- Needs to be usable – concise form, right concepts that are the same or map to receiver’s concepts.
- Delivered at the right time – or available at the right time.
- Need to suit the means of delivery: communication, reporting, documenting.
Characteristics of good metrics:

- Interesting.
- Directly map to someone’s quality related value (defect, risk, what money is made from). Make sense: engineering sense or business sense.
- Give a common talking point.
- Simple.
- Easy to create.
- Difficult to manipulate.
- Will not cause manipulation if set a goal (like adding bad test cases if the amount of passed test cases is a goal).

Characteristics of a good test report:

- Immediately shows the overall situations.
- Designed to support someone’s decision making. Shows data that relates to the decisions, using concepts that are meaningful in the decision making, for the decision maker.
- Different stakeholder with different needs should get different reports.
- Suits the company’s culture in design, attitude.
- Includes tester’s assessment, not just numbers.
- Clearly and strongly shows areas that most urgently need improvement.
- Is interesting and pleasant to read.
- Is layered – short overall view allows going deeper into the data.
- Most of it created automatically.
- Is always available.
- Intranet / extranet reports work with mobile devices.
- Supports all needs of product, quality and risk management.
5. Metrics usage mind maps

Figure 3. Different aspects of using empirical data and metrics.
Figure 4. Product metrics usage.
6. Challenges and trends with metrics and reports

- People measure what is easy to measure. Pass rate for test cases is a traditional metric, but what does it tell? Does the number of test cases tell something? Big numbers always look good, but what is the real situation.

- New test technologies (such as MBT) are developed by people who are too attached to the internal metrics of their system, but external metrics are needed for decision making.

- Testing scientists and engineers aim for exactness, but business decision making is always fuzzy. Non-exact information that makes people understand issues is more important than exact information that does not.

- While it is good to create metrics automatically, not all need to be created that way. Testers can synthetize higher level metrics using his/her intelligent mind. When there is an opportunity to think, it should be used!

- Many experts emphasize tester’s feeling about the situation both regarding to “has something been tested properly” and “is something ok based on tests”.

- There is a trend of using colour in reporting – green or blue, yellow, red.

- Many are using mind maps in reporting – as well as test design.

7. The culture of “done”

The agile development culture often neglects metric other than “task logistics” – is something done or not? That applies to testing too. When a new feature has been tested, the tester makes the decision (based on personal feeling or perhaps low-level metrics) that the testing for that parts is declared as done and the feature can either be declared as “done” (after the corrections of course) or can move to a next workflow step. That reduces low-level metrics to something more personal, and the business level metrics to just information about some items having passed the development pipeline.

That is the simplest way to do things. Of course organizations can add levels and phases above and after that as needed, where the metrics usage is different. For example, MBT may have a role not in the team, but as a simultaneous testing process (continuous or rhythmic) and its coverage and other metrics can and should often be more traditional. (Yet we need to remember, that traditional in practice does not always refer to something that has been a living culture, but something that should have been a living culture.)
8. **Metrics on various domains of activity**

Basic idea: defect related metrics depend on what the aim of testing is. Often we are in some clear “working domain”. Note: this applies to only functional characteristics of the system.

*Table 1. Some metrics at various "domains of activity".*

<table>
<thead>
<tr>
<th>Performance metrics</th>
<th>Maturity domain</th>
<th>Reliability domain</th>
<th>Customer acceptance domain (depends on system)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defect finding domain</td>
<td>Defect count</td>
<td>Defect count</td>
<td>MTBF (accelerated)</td>
</tr>
<tr>
<td>By seriousness</td>
<td>Defect count trend</td>
<td>Failure rate</td>
<td>Defect count</td>
</tr>
<tr>
<td>By priority</td>
<td>Defect density</td>
<td>Action success rate</td>
<td>By use case</td>
</tr>
<tr>
<td>By risk</td>
<td></td>
<td></td>
<td>By risk</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>External control metrics</th>
<th>Maturity domain</th>
<th>Reliability domain</th>
<th>Customer acceptance domain (depends on system)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coverage</td>
<td>Coverage</td>
<td>Coverage</td>
<td>Coverage</td>
</tr>
<tr>
<td>- Requirement</td>
<td>- Requirement</td>
<td>- Requirement</td>
<td>- Requirement</td>
</tr>
<tr>
<td>- Functional</td>
<td>- Functional</td>
<td>- Functional</td>
<td>- Functional</td>
</tr>
<tr>
<td>- Keyword (high level)</td>
<td>- Keyword (high level)</td>
<td>- Keyword (high level)</td>
<td>- Keyword (high level)</td>
</tr>
<tr>
<td>- By component</td>
<td>- By component</td>
<td>- By component</td>
<td>- By component</td>
</tr>
<tr>
<td>- Use case / user story</td>
<td>- Use case / user story</td>
<td>- Use case / user story</td>
<td>- Use case / user story</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Internal control metrics</th>
<th>Maturity domain</th>
<th>Reliability domain</th>
<th>Customer acceptance domain (depends on system)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coverage</td>
<td>Coverage</td>
<td>Coverage</td>
<td>Coverage</td>
</tr>
<tr>
<td>- Code</td>
<td>- Code</td>
<td>- Code</td>
<td>- Code</td>
</tr>
<tr>
<td>- State</td>
<td>- State</td>
<td>- State</td>
<td>- State</td>
</tr>
<tr>
<td>- State transition</td>
<td>- State transition</td>
<td>- State transition</td>
<td>- State transition</td>
</tr>
<tr>
<td>- Application switch</td>
<td>- Application switch</td>
<td>- Application switch</td>
<td>- Application switch</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Goals</th>
<th>Maturity domain</th>
<th>Reliability domain</th>
<th>Customer acceptance domain (depends on system)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internal standards</td>
<td>Internal standards</td>
<td>Internal standards</td>
<td>Development contract</td>
</tr>
<tr>
<td>Requirements</td>
<td>Requirements</td>
<td>Requirements</td>
<td>Requirement specification</td>
</tr>
<tr>
<td>Quality plan</td>
<td>Quality plan</td>
<td>Quality plan</td>
<td>Usual accepted level</td>
</tr>
</tbody>
</table>
9. Lessons for model-based testing

- Nobody really cares (should care) about models, paths and states and such – quality is the main thing. How do things work for their purpose? What is the value of the testing – what does it reveal?
- Metrics are not all-purpose.
- Internal coverage metrics are important for those who develop models and testers who do low-level testing.
  - Path coverage, state coverage, transition coverage, action word coverage, action-state proposition coverage etc.
- But more important that coverage, is knowing, what areas have been covered and what not.
  - Visual mark-up may help in seeing the picture.
- Good testing is already focused and that reflects to the metrics.
- For product-level decision making, external coverage metrics are more useful.
  - Requirement coverage.
  - Functional coverage.
  - Data domain coverage.
  - Interaction coverage, including application switch coverage.
- MBT needs to show its unique value and focus, and when it is focused to something like testing interactions, the metrics should support that focusing.
- While it is good to create metrics automatically, not all need to be created that way. Testers can synthesize higher level metrics using his/her intelligent mind.
- Tester’s subjective feeling is important to include in reports.
  - Overall situation (green, yellow, red).
  - Situation in product areas.

10. What is different with robots?

A testing robot is just a tool so it should not have any influence on the product quality metrics. But robots are expensive workflow tools, so we should be interested in how well the investment is used. We have not even discussed any workflow metrics yet, but metrics such as this should be interested:

- Percentile of time the robot arm, finger is in action (and not waiting for, for example, the test system to do image processing).
- Percentile of time the robot is doing test runs and not just sitting idle waiting for a test to start.
- Mean time between failures (MTBF), divided into software and hardware failures.
- Availability. The time the system can be used for testing = (total time – time spent in failures and on preventive maintenance) / total time.
11. References


APPENDIX: Some knowledge needs that testing should give information to:

To the customer (of a tailored information system):

- Can we accept the delivery and pay the provider?
- Has the provider done their testing as promised?
- If we start using this system, will things work ok? (For end users, internally)
- Are the some business processes / features that will not work ok? What are the problems in them?
- Are all business processes tested?
- Are all end user use cases tested?
- Can we expect that our personnel will accept and even like the new system?
- Will the transition to the new system be easy?
- Will the new system handle the load of heavy usage?
- If we launch the new system, will we be in the news tomorrow because of problems?
- Will the new system work with our other 250 systems?
- Will the new system work with old data, from past 40 years?
- Is the new system secure?
- Etc...
To **product ownership / management:**

- Can we ship? Is the product mature enough to ship?
- Do the features we have promised to work actually work (well enough)?
- Do the new exciting features (new technologies that we advertise) work great?
- Can we expect that customers will be satisfied & will not complain?
- Can we expect that customers will not need to call our support too often?
- If we ship, can I expect any trouble?
- What are the main risks, risky areas?
- Are there known hazards?
- How much effort have we put into testing?
- How can I prove to customer that the testing has been proper & according to contract?
- Have we done everything so that standards are satisfied (like safety standards)?
- How can I prove to government(s) that testing has been proper & meets laws & standards?
- How can I prove in court that testing has been proper (in case of problems)?
- Is there some quality related information that I should know about?
- Etc...

To **project management / project quality management:**

- How mature is the product?
- What is the trend of maturing?
- When can we estimate it can ship?
- Is the current build solid enough for system / alpha / beta / customer testing?
- Which build is solid enough for system / alpha / beta / customer testing?
- What is the testers' opinion on the product's quality?
- If we ship, can I or the team expect any trouble?
- What components are not yet good enough?
- Do some feature / component / area have severe problems?
- Does some team or some developer have severe problems?
- Do we improve the product or move backward?
- What defects are there?
- What defects should be fixed?
- What defects should be fixed first?
- Has everything relevant been tested (covered)? (Requirements, features, use cases / user stories, interactions, environments...)
- Have the critical areas been tested properly?
• Does the testing meet requirements (like safety standards)?
• Has the team done a good, professional job?
• How can I prove to product management (or customer) that the testing has been proper?
• Etc...

**To a developer:**

• Are my components ok?
• What defects are there? How serious are they?
• What defects should be fixed?
• What defects should be fixed first?
• Where are the defects (as exactly as possible)? How can I repeat them, to fix them? What environments, conditions do they apply?
• Have I done a good, professional job?
• (Plus many of the things others need to know -- but many of them as "nice to know")
• Etc...