Risk planning for testing

Hans Schaefer
hans.schaefer@ieee.org

Project risks for the Tester
Prioritization of a first test
Prioritization of the later tests
Traditional project management focuses on project risks rather than product risks, but there are also project risks for testing. Some examples of project risk are:

- **Timing issues**
  - Delays are a very common cause of trouble for testing. This means that a test project always has to take into account that the software to be tested is not supplied to test execution at the planned point of time and that less time will be available. There should always be a plan for prioritizing tests; there should be plans for alternative orders of testing, more parallel testing, use of more platforms, third parties, or even testing after delivery.

- **Supplier issues**
  - Failure of a third party who should supply deliverable, i.e. software modules, components, subsystems or libraries, or tools for development or testing. There may also be problems with supply of third party testing services.
  - Contractual issues, i.e. badly formulated or followed up contracts, resulting in late, wrong or no deliverable at too high costs.

- **Organizational factors**
  - Skill and staff shortages, lack of specialists. This may mean no help is available when testers need consulting about the platform and environment.
  - Personal and training issues, i.e. Availability of the wrong people without adequate knowledge. Testing has to use any low qualified people just available at the time.
  - Political issues, i.e. Problems to define the right requirements, or problems for the testers to communicate their needs and test results, failure to follow up on low level testing and reviews. This may also mean the people available do not have the right attitude to testing.

- **Specialist issues**
  - Feasibility: The extent that requirements can be met given existing constraints. How near to the state of the art is the project, how much new to the organization?
  - The quality of the design, development and test team. Low quality teams may produce low quality software, which increases the cost of testing. Low quality test teams do not find the defects, resulting in increased risk after testing.
Risks BEFORE Testing

Bad Quality
Many faults overlooked
Blocking faults
Too many new versions

-> Requirements to, and follow up of quality assurance before test

Delays
-> Alternative Plans

Lack of knowledge
-> Test of earlier Versions, early test planning, involvement in design
Risks AFTER Testing

THESE SHOULD NOT HAPPEN…

Customer finds faults.
Customer uses the product in new ways.

Analysis of necessary reliability!
Risks in the Test project itself

Bad management
Lack of qualification
Too few or the wrong people, too late
Bad coordination
Bad cooperation
Problems with equipment and tools
Simulators, scaffolding, utilities take time to get

Medicine: Normal good project management.
How to make testing cheaper?

Good people save time and money
Good prioritization

Try to get rid of part of the task...
Getting rid of work

Get someone else to pay for it or cut it out completely!
  – Who pays for unit testing?
  – What about test entry criteria?
  – Less documentation

Cutting installation cost - strategies for defect repair
  – When to correct a defect, when not?
  – Rule 1: Repair only defects causing important failures (blocking the tests)!
  – Rule 2: Change requests to next release!
  – Rule 3: Install corrections in groups!
  – Rule 4: Daily build!

Less Test, should the customers pay ????
Risk based Test - Practice

For First Test: Identify what is critical

1. "Top-10"

Test identifies areas with lots of detects

2.

For Second Test: Extra Testing:
- Extra Test by product specialist
- automated regression test

3.
Prioritization for the first test
Product Risks: What to think about

- Which functions and attributes are critical?
  - (essential for the business success to reduce the business risk).
- How visible is a problem in a function or attribute? (for users, customers, their users or customers, people outside, licensing agencies)
- How often is a function used?
- Can we do without?
- The cost of damage
- Legal consequences

Identify Areas of “High Risk”
Probability and Cost
All functions/modules should be tested to a “minimum level”
“Extra Testing” in high risk areas
Establish Test Plan and Schedule
Monitor Quality
  Number of Faults per function and time
Monitor Progress
  Number of hours in test and fix -> ETC
Failure probability: What is (presumably) worst?

Choose the three most important of this!

- Complex areas
- Changed areas
- Number of people involved
- Turnover
- New technology, solutions
- Time pressure
- Areas which needed optimizing
- Areas with many defects before
- Geographical spread
- History of prior use

Always ask several people!
Ask several times!
Document your rationale!
Complex - even simple areas may fail!


In this article, they use metrics to predict fault-prone modules and show how testing can be directed in order to save resources.
Do not forget

Can we test ONLY PART of the product?

Other versions later?

Fight time pressure!
How to calculate priority of risk areas?

Assign weights to the chosen factors. (1 - 3 - 10)
Assign points to every area and factor
(1 - 2 - 3 - 4 - 5)
Calculate the weighted sum (impact * probability).
The spreadsheet does not contain the “surprise” factor, but that can be added.

Spreadsheet
Download: http://home.c2i.net/schaefer/testing/riskcalc.hqx
**Example**

<table>
<thead>
<tr>
<th>Area to test</th>
<th>Damage</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Usage frequency</td>
<td>Visibility</td>
</tr>
<tr>
<td>Weight</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>Function A</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Function A performance</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Function B</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>F B usability</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Function C</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Function D</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

In this example, it has been assumed that the functional volume of every area is equal.
What is the formula?

Risk = Impact * Probability

Impact =  
(Weight for impact factor 1 * value for this factor +  
Weight for impact factor 2 * value for this factor + + +  
Weight for impact factor n * value for this factor )

Probability =  
((Weight for probability factor 1 * value for this factor +  
Weight for probability factor 2 * value for this factor + + +  
Weight for probability factor n * value for this factor )) /  
(functional volume)

In the example, I set all functional volumes to 1, i.e. Equal. Functional volume shows how much functionality a function or test area contains. It is not complexity, it is rather the amount of work put into the implementation. (Complexity is one of the probability factors?)
The mathematics behind it

It works well enough.
We may actually be on a logarithmic scale (humans assigning points do so), which means we should ADD instead of MULTIPLY.
The highest weighted sums -> thorough testing
Middle weighted sums -> ordinary testing
Low weighted sums -> light testing

Make sure you use your head! Analyze unexpected results!
What to do if you do not know anything about the product?

Run a test.
Prioritize roughly by risk.

First a breadth test ("smoke test"), everything a little, risky items more.
Then prioritize a more thorough test for the second test cycle.
Prioritization of further test cycles

Adaptive testing

Fault- and Coverage analysis
Analysis of defect detection percentage
Analysis of test coverage

Have all (important) functions been covered?
Exception handling?
States and transitions?
Important non functional requirements?

Is test coverage as planned?

Extra Check or Test where coverage differs from expected coverage!
How to analyze your test

Coverage against expected coverage

Is the code coverage under test as expected?

If some area is executed a lot more than expected, is that a symptom for performance problems? Bottleneck, error?

If an area was covered less than expected, is that area superfluous, or was the specification too “thin”?

Do an extra inspection of such areas!
Analysis of fault density

Facts:

Testing does not find all faults.
The more you find, the more are left.
Post-release fault density correlates with test fault density!

Defect prone units:
- A Pareto distribution.
  NSA: 90% of high severity failures come from 2.5% of the units.
  Others: Typically 80% failures from 20% of the units.

Defects are social creatures, they tend to keep together!
What to use fault density for

- Measure the number of faults / 1000 lines of code.
- Compare with your own average.
- Spend extra analysis or test if the program under test is bad.
- Spend extra analysis if the program under test is “too good”.

Error density, best values:

From the Pacific Northwest Software Quality Conference in Portland Oregon on October 28 and 29, 1997:

George Yamamura of Boeing gave a world class talk describing how Boeing's Space and Defense System's Division achieved a level 5 CMM. He reported a rate of 5.3 defects introduced for every 100 lines changed and showed how using this data Boeing reduced the number of defects in their final product by catching errors earlier in their development process. He found defects highly correlated with personnel practices. Groups with 10 or more tasks and people with 3 or more independent activities tended to introduce more defects into the final product than those more focused on the job at hand. He pointed out that large changes were more error prone than small ones, with changes of 100 words of memory or larger being considered large ones. The most startling data is the 0.918 correlation between defects and personnel turnover rates.
Example by Lars Wahlberg, OM Group, Stockholm:

The results can be interpreted by comparing the defect and test case bars. If the defect content is higher, it means that there are more defects, compared to tests executed. For an example, Quotes has about 120% more defects than average, but only 30% more test cases. Problem areas would then be Quotes, Trade Reporting, DTCC and NSIP (minor).

The functional complexity of DTCC was very low (about 0.4), which makes it sensitive to variations in number of defects or number of test cases. The real number of defects found on DTCC is only 3, which is a too low a number to make any decisions on. It is judged that there aren’t enough defects to highlight DTCC as a problem area.

Conclusion: More test should be done for Quotes, Trade reporting and NSIP.
Analysis of causes

If you have many defects with the same cause category, think about improving your way of working!

Typical for unit testing: (Classification see IEEE Std 1044)
- Logic
- Computation
- Interfacing
- Data handling
- Input data problem
- Documentation
- Change
Analysis of defect detection

How effective is the already planned or done defect detection?
Or: How much chance is there that defects survive?

Probability for defects decreases:

New risk = old risk / detection percentage

Defect detection percentage = defects found / defects before detection
measure * 100%
Test reporting, risks and benefits

Diagram:
- **Testing** addresses **Risks**
- **Benefits** demonstrates **Project status**
- **Risks** threaten **Benefits**
- **Benefits** inform about **Project status**
The following risks have been tested: ….
- we can regard the risks as addressed and more or less forget about them
- New and better project status

The following risks are still not passed:
- The risk is still of concern
- No change in project status
- We have more knowledge of projects status (maybe delays become inevitable)

The following risks have been reduced: (Change color).
If tests pass:

- We can demonstrate that certain benefits are not realized
- There is visible real progress
- A change of status in the project occurs

If tests fail:

- Failures for high benefits are of high priority
- Clear priorities.

Some risks may come from later THAN START!!!!!
References

IEEE Standard 1044-2002: Standard Classification for Software Anomalies
-You find these standards at sales@ieee.org

Rex Black, Managing the Testing Process, John Wiley, 2002. (includes CD with a test priority spreadsheet)

http://home.c2i.net/schaefer/testing/risktest.doc
James Bach, Risk Based Testing, STQEMagazine, Vo1, No. 6,
www.stqemagazine.com/featured.asp?stamp=1129128440
FMEA: Failure Mode and Effects Analysis: http://www.fmeainfocentre.com/