5th World Congress for Software Quality – Shanghai, China – November 2011

Useful Metrics for Managing Testing

Tim Brister
Jaaloh Ltd, 47 Colney Lane,
Norwich NR4 7RG, England
Timjbrister@aol.com

Karol Frühauf
INFOGEM AG, Postfach,
5401 Baden, Switzerland
Karol.Fruehauf@infogem.ch

Dr. Ferdinand Gramsamer
bbv Software Services AG,
6002 Luzern, Switzerland
Ferdinand.Gramsamer@bbv.ch

Abstract

The paper presents a couple of test execution and defect report metrics that were useful in managing a software project with ca. 40 testers. The metrics were used to recognise the need for actions and for deciding when to launch the software. It is nothing spectacular we report on except the fact that the figures were continuously evaluated and turned out to be useful and instructive. Therefore we focus this experience report on lessons learned. We omit any hint on the application area because we're convinced that it is not a factor for the usefulness of the presented metrics.

1. Introduction

The major challenge in test management, besides the human factors, is tracking the test execution progress and to forecast when the product will be ready for launch. We report here on a few simple metrics that were useful for these purposes in a project with a four level testing strategy: component test, integration test, system test, and business and user acceptance test.

For test management three classes of metrics are of interest:

- Test case design and specification related metrics
- Test case execution related metrics
- Defect related metrics

In our project the majority of the test specifications (TSPC) were inherited from the past. New test specifications on the lower testing levels were partly derived from already existing ones. Therefore the first class of metrics was of less relevance; the progress of the test design activities was tracked by the status of the new / updated test specifications.

We focus here on the metrics around test execution and defect removal and report mainly about their interpretation.

No cost data will be reported. We will not provide a description of the data acquisition process. We would like to stress that the effort dedicated exclusively to the metrics work was negligible compared with the overall test management cost alone.

The metrics selection and its usefulness are independent of the application area, which is why we abstain from presenting the application.

2. Goals and Basic Metrics

2.1 Goals for the Evaluations

The underlying planning assumption was that test execution is prioritised based on risks, therefore wherever possible the area of highest risk is scheduled to be tested earliest and most often.

Project management wanted to know

- which functional and non-functional aspects have been tested, at which test level and how many times?
- Which functional and non-functional aspects have not been tested yet?
- What is the schedule for
  - regression test of already tested aspects?
  - test of the yet untested aspects?
- How many critical defects

- were found in the reporting period?
- were closed in the reporting period?
- are still open at the end of the reporting period?

- When can we launch?

Additionally, test management needed to quickly pinpoint issues or incidents. The concept of test specifications and test areas meant that the functional and non-functional aspects could be consolidated for the purpose of maintaining an overview whilst also enabling test management to refine information and report on the aspects relevant to the particular issue or incident.

2.2 Basis of the Evaluations

In the beginning of the project, we have defined test execution metrics that would serve two means:

1. Respect the needs of testers
2. Provide meaningful results to project management

The ultimate requirement of the testers is: No additional effort for metrics data. This has been achieved quite easily; the established test execution process at system test level already captured the data needed for management purposes and was therefore sufficient for the evaluations we introduced for management. For the other test levels a similar process was put in place.

At each test level (component test, integration test, and system test) the test scope has been broken down into several test specifications. The topic of a test specification is the functional or non-functional behaviour of a certain feature or functionality provided by a particular interface of an entity.

Each test specification is organized in test areas, which are compounds of test cases that logically belong together. For instance, for a router, the routing would be a test area, the administration GUI another one, etc. As shown in Table 3 there are at system test level 23 test specifications, adding up to 125 test areas and totalling in 1642 test cases.

A test case is a series of test steps. Testers record their progress and the results of manual test execution with the granularity of test steps. The test steps specify the checks for test automation.
Progress on test step level was monitored by the tester and the leader of his / her test team. Project and test management monitored the state and results of test cases, test areas and test specifications.

Test management reported test progress on a per test case basis cumulated across all test areas per test level. Figure 2 shows the progress per release for component A, Figure 1 the progress for a system test cycle. We were able to provide periodically an overview which functional and non-functional aspects have been tested, how many times this happened and what were the results (see also Table 1 and Table 3).

In the evaluations we distinguished between the state of a test case / test area and the result of the execution. Depending on test execution progress a test case can have different states.

State	Description
not yet executed	as long as no test step was yet executed (0% test steps executed)
not completed	test case execution started but not all test steps are executed yet (0% < test steps executed < 100%)
completed	100% test steps executed
not planned	none of the test steps is planned to be executed

Additionally, a compound test case state “not run” was introduced to indicate that all of its test steps are either “not planned” or “not yet executed”.

The same states are defined for a test area. A test area is “not planned” when none of its test cases are planned to be executed and is “completed” if all of its test cases have been executed.

The evaluation of the test areas executed can deliver the following results:

- “passed” all test cases passed
- “failed” at least one test case failed.

- "failing" not completed yet but at least one executed test case "failed"
- "not all run" not completed yet but all executed test cases "passed"

Other useful states were "out of scope" and "error". "Out of scope" is a test area that is no longer relevant to the project but was not removed from the test specification yet. A test area will end in a state "error", if the evaluation of the result fails. The latter two states were in particular useful to more easily manage the test specification changes that had an impact on the metric data acquisition process, and hence we will not further discuss them.

The defect removal workflow was already defined and supported by a tool. Only the recording of the test level, at which a defect is detected, had to be amended and the evaluations for project and test management had to be prepared.

The following attributes of defect reports were evaluated:

- Test level at which the defect was detected
- Component affected by the defect
- Severity of the defect
- Date the defect report was submitted
- Date the defect report was closed

A defect tracking tool was used and its output processed by a spreadsheet tool.

Early in the project it was decided that only critical defects, i.e. those that prevent useful operation of the system, would be repaired. Therefore monitoring and reporting of defects was focussed on these.

Data acquisition and preparation was done on a weekly basis and did cost about 1h per integration and system test level and one hour per component. The consolidation by overall test management took 4h. All in all the measurement effort was around 12h per week, i.e. less than 1% of the available testing effort.

3. Test Case Execution Metrics

3.1 Test Progress – Specified, Planned, and Executed Number of Test Cases

For each release, the number of test cases specified and planned for execution was evaluated per component and test level. This was done to set clear goals for testing, so we could measure success and hence make a statement about readiness for launch. The time-period for testing a release was four weeks.

We made test execution progress visible on a weekly basis as shown in Figure 1 for functional system test of release V1.10. To prevent overload of this graph, the results "passed", "failed" are displayed together with the states "not completed" and "not run" as a stacked bar, and the "planned" number of test cases as a line. The benefit is that we always see the total number of specified test cases (the entire bar), the number of test cases "not planned", which is the difference between the top of the bar and the line, and the number of test cases "not yet executed", which is the number of test cases "not run" below the "planned" line.

As can be seen in Figure 1 most test cases were executed during the first week. The number of additional test cases executed per week goes down from 36% in first week, to 18% in last week in this particular release. We observed the same pattern for all other releases on all test-levels. At system level a significant portion of test cases is executed manually, which allows for the state "not completed". It signifies that a tester found issues while executing a test that need to be resolved until end of the cycle. The number of "not completed" is to this end an expression of the maturity of the software. The less the number of "not completed" test cases the higher the software maturity. The ultimate goal is to have zero test cases in state "not completed", which we achieved in the final release. The planned number of test cases decreases slightly; it is modified based on insights derived from the executed tests.

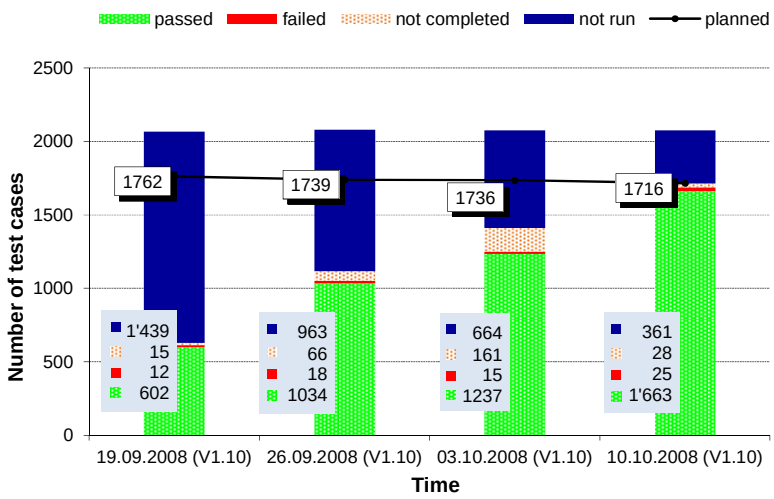
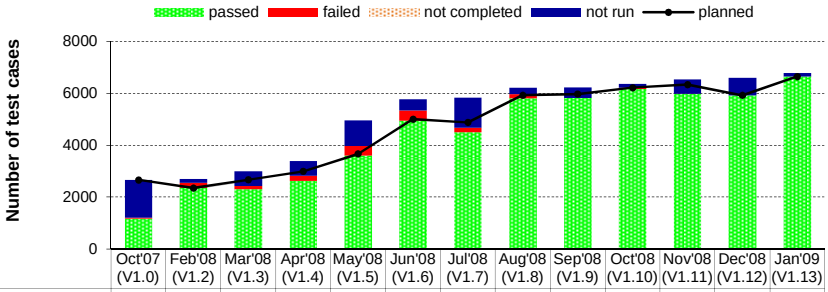


Figure 1. Weekly test-execution progress of release V1.10 of system tests

Whereas Figure 1 shows intra-release progress, Figure 2 demonstrates inter-release progress. It compares the number of executed test cases for each result category with the planned ones over the releases, here for component A.



	Oct'07 (V1.0)	Feb'08 (V1.2)	Mar'08 (V1.3)	Apr'08 (V1.4)	May'08 (V1.5)	Jun'08 (V1.6)	Jul'08 (V1.7)	Aug'08 (V1.8)	Sep'08 (V1.9)	Oct'08 (V1.10)	Nov'08 (V1.11)	Dec'08 (V1.12)	Jan'09 (V1.13)
not run	1455	143	576	584	985	435	1157	262	410	152	546	685	140
not completed	0	0	0	0	0	0	0	0	0	0	0	0	0
failed	28	204	128	184	375	397	177	151	0	44	3	0	0
passed	1177	2351	2300	2634	3598	4952	4502	5816	5822	6175	5986	5925	6655
planned	2660	2351	2669	2994	3665	5009	4879	5930	5974	6221	6343	5925	6655

Figure 2. Success of test-execution per release of component A across releases

The first test cycle (V1.0) showed a significant amount of test cases that were “not yet executed”, indicated by the number of “not run” below the “planned” test cases. This let us conclude that the component software was not yet ready for integration and system test. This information helped us to postpone start of system test to V1.2. There was no component test for V1.1 because component A deliverable did not pass acceptance testing; the component test cases were used to help development.
The functionality of the component A did grow with the releases and so did the number of specified and planned test cases. It was at V1.11 stage that all functionality was implemented. As component A was considered high risk, almost all test cases were regressed for all releases.

3.2 Test Coverage – Test Execution and Results History

By keeping a history of the test results of a release per test level and test area, we were able to answer the following questions:

a) Are all test areas covered?
b) Is the test execution coverage of all test specifications 100%?
c) How often has a test area been executed?

d) What was the last time, a test area has successfully been executed?

The information that we kept were the states of a test area as described in Section 2.2. Furthermore, we tracked the number of specified test cases, how many were executed and how many "passed". This information was sufficient to derive a full picture and answer the above questions.

As an example in Table 1, you see test specification A01, with the test areas A, B, C, and D. Test areas A, B, and C passed, because the number of specified test cases were all executed and passed. Test area D is "not all run", because all executed test cases passed, but not all were run. Similarly, for test specification A02, test area A is failing, because not all executed test cases passed, but not all are completed already, and test area C failed, because all specified test cases were executed, but 4 test cases failed. With this table we are able to answer the questions a) and b).

When putting this kind of information in a sequence per release from left to right, one can easily derive how many times a test area has been executed, when it was last successfully executed and whether or not all test areas or test specifications were covered. This is shown in Table 2 for the releases V1.8 to V1.12 for test specification B23. The states are abbreviated In this table: "p" stands for "passed", "nr" signifies "not run", "pl" means "planned" and "np" "not planned". The last two columns summarize all releases to show, whether a test area has so far **not** been covered, indicated by a 1 in the second last column, or how many times it has been executed – shown in the last column.

In the end, we were able to pull all the information about test coverage together in one table to answer questions a) – d), see Table 3 for system test level. There are three blocks of information. The left one gives an overview about the available test specifications and test areas. The block in the middle shows the results of the current release, whereas the block on the right summarizes the history.

Specifically, Table 3 summarizes release V1.13, which is the release that went into production. We can see that test specifications A03 and A04 were not planned for execution. Hence the percentage of test execution is 0%. However, the test specification has been fully covered and executed once before. A03 and A04 were low priority test specifications. The focus goes quickly to those test specifications of interest: B23, and C02. B23 contains a test area covering a feature that was always promised, but after all has never been delivered. C02 is also a low priority test specification. It contains a test area that was, because of its destructive nature, difficult to execute and would have blocked the test environment for at least one day. Based on this information project management therefore decided not to execute these tests and take the risk.

Table 1. Test area result overvierw

Doc Id	Test Area		Test Cases		
Name	Name	State	Specified	Executed	Passed
A01	Test Area A01.A	Passed	10	10	10
	Test Area A01.B	Passed	21	21	21
	Test Area A01.C	Passed	5	5	5
	Test Area A01.D	not all run	15	10	10
A02	Test Area A02.A	Failing	703	634	632
	Test Area A02.B	not planned	0	0	0
	Test Area A02.C	**Failed**	**34**	**34**	**30**
	Test Area A02.D	not planned	1	0	0
	Test Area A02.E	Passed	115	115	115
	Test Area A02.F	Passed	120	120	120
	Test Area A02.G	Failing	132	127	125

Table 2. Test area results across consecutive releases

Name		Status					since 1.8	
TSPC	Test Area	V1.8	V1.9	V1.10	V1.11	V1.12	Test Areas Not Fully Covered	Nr Times Executed
B23	Test Area B23.A	p	pl	p	pl	p	0	3
B23	Test Area B23.B	p	pl	p	nr	p	0	4
B23	Test Area B23.C	p	pl	p	nr	p	0	4
B23	Test Area B23.D	p	np	p	pl	p	0	3
B23	…	…	…	…	…	…	…	…
B23	Test Area B23.U	nr	pl	p	pl	p	0	3
B23	Test Area B23.W	p	pl	nr	nr	p	0	4
B23	Test Area B23.X	np	pl	np	np	np	1	0
B23	Test Area B23.Y	p	pl	nr	nr	p	0	4
B23	Test Area B23.Z	p	pl	p	pl	p	0	3

The advantage of that table is that it is easy to recognise what has been done and what not, thus is a good basis to decide with project management the measures and appropriate action. Only the exceptions will be discussed.

Table 3. Test area coverage and frequency of execution

Specified			Release V1.13					Since V1.8	
TSPC	Priority	Number of Test Areas	Passed	Failed	Not Completed	Not run	Percent Executed	Test Areas Not Fully Covered	Nr Times Executed
A01	2	8	126	0	0	0	100.00%	0	4
A02	2	1	8	0	0	0	100.00%	0	4
A03	3	1	0	0	0	0	0.00%	0	1
A04	3	2	0	0	0	0	0.00%	0	1
B01	2	4	62	0	0	0	100.00%	0	4
B11	1	14	934	0	0	0	100.00%	0	2
B21	2	5	27	0	0	0	100.00%	0	3
B22	1	3	3	0	0	0	100.00%	0	4
B23	1	26	133	0	0	0	100.00%	1	3
…	…	…	…	…	…	…	…	…	…
C01	1	2	1	0	0	0	100.00%	0	3
C02	3	8	0	0	0	0	0.00%	1	1
C03	2	2	4	0	0	0	100.00%	0	3
23		**125**	**1642**	**0**	**0**	**0**		**2**	

4. Defect Metrics

The following metrics will be discussed:

- Cumulated number of submitted, closed and open critical defect reports
- Cumulated number of submitted defect reports per component
- Monthly rate of submitted defect reports per test level
- Survival period of critical defect reports

4.1 Submitted, Closed, and yet Open Defect Reports

The shape of the number of reported critical defects in Figure 3 is no surprise, it is as everybody would expect. The number of closed defect reports is tricky to understand. Remember, we have three test levels. We considered the defects that already passed at least one test level as potentially ready for launch. Therefore these were added to the number of closed defect reports. The number of open defect reports counts all those that are not closed. Therefore, the number of open defect reports is not the difference between the number of submitted defect reports and the closed ones.

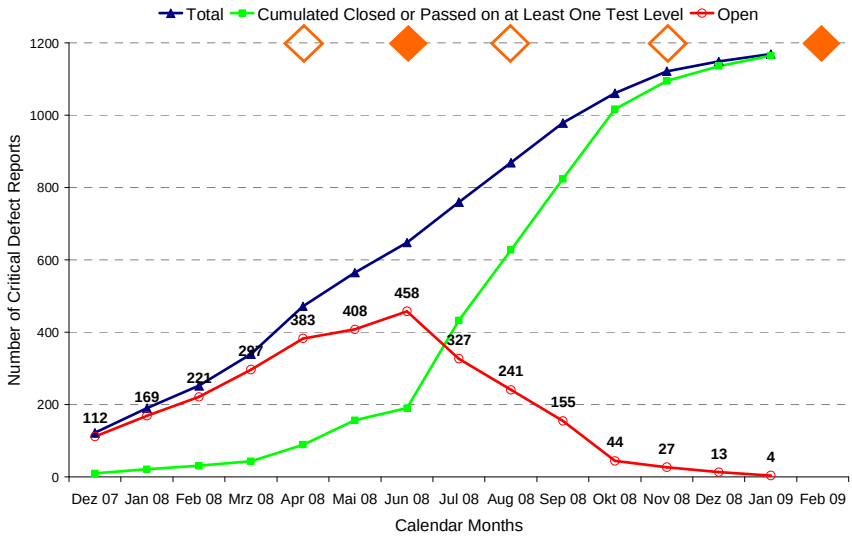


Figure 3. Cumulated number of critical defect reports

To find out whether the development team needs to be inspired or the test team is the bottleneck, a list of critical defect reports "in work" and "in test" was produced. Project management encouraged the development team and test management the test team to accomplish the work in time.

The original internal plan was to launch the product End of June 2008 (full rhomb in Figure 3). The launch date announcement was scheduled for April 2008 (empty rhomb in Figure 3). At that time a new announcement date was published for August 2008 and a tentative launch date for the first quarter 2009. In August 2008 the launch date in February 2009 was promised with a confirmation due in November 2008. The launch date in February 2009 was confirmed in November.

The number of submitted critical defect reports curve in May could have been interpreted as entering the saturation phase. Fortunately, the forecast had to be done in April where this was evidently not the case. In April there was a peak with a record for the number of submitted reports in a month (see Figure 5 that shows this peak more clearly). In August the number of open critical defect reports supported the announcement, closing of defect reports outperformed their submission. In November the projection of both curves (not shown in Figure 3) indicated that the launch date in February 2009 is feasible.

It is worth mentioning that the last functional extension arrived in November. This explains the nearly constant defect rate in previous periods. As soon as only defect removals were delivered, the system started to stabilise. This was considered in the projections we've made for the launch date forecast and

indeed, the number of induced defects was low. In the last 6 months it was as low as 3%. Till September 2008 yet around 12% of removed defect tests failed.

Did all components behave similarly? According to Figure 4 the components A and B were in November 2008 still at risk while C was doing well.

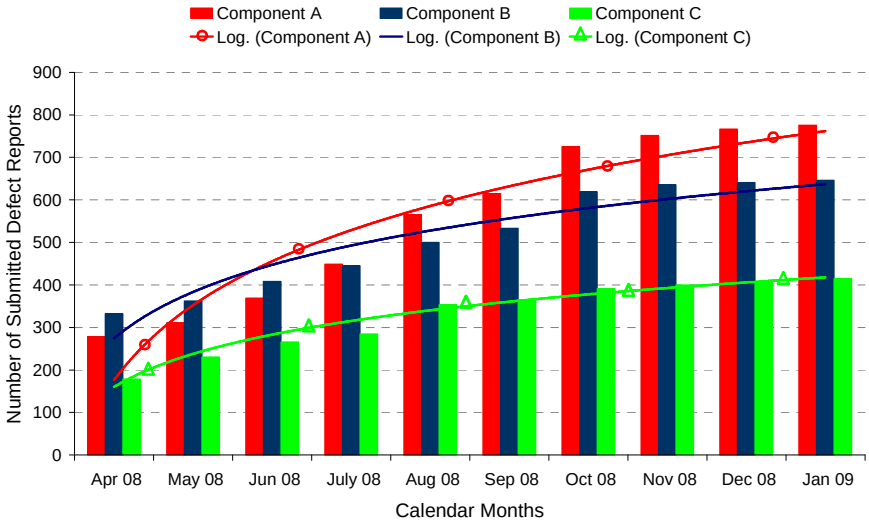


Figure 4. Cumulated number of submitted defect reports per component

The figure contains only the data for the critical three components that were in focus of monitoring. All three components had at least one release that was a major instability step. These were caused by major functional enhancements in components A (August and October) and C (August). Component B experienced in October a significant redesign within its core functionality. Component C became quicker mature because the overall functionality was implemented quite early. No surprise, it is the smallest component.

4.2 Defects Detected per Test Level

The defects were assigned to the test level in which they were detected. As expected, most of the defects were found at component level that started far ahead of integration and system test as can be seen in Figure 5. The explanation for the integration test's low defect detection rate is that only few interfaces were tested formally and that the test case design was done partly with end-to-end test in mind and not focused on the interfaces between the components.

4.3 Defect Report Survival Period

The defect reports issued early in the project took a long time until they were fixed and got closed (see Figure 6). The survival time of the defect reports did not drop below 50 days until September (6 months before launch). With the more or less regular monthly release cycle a survival time of 45 days would be a good practice. As expected the survival period dropped below the monthly cycle period at the end where defect fixes were delivered in patches.

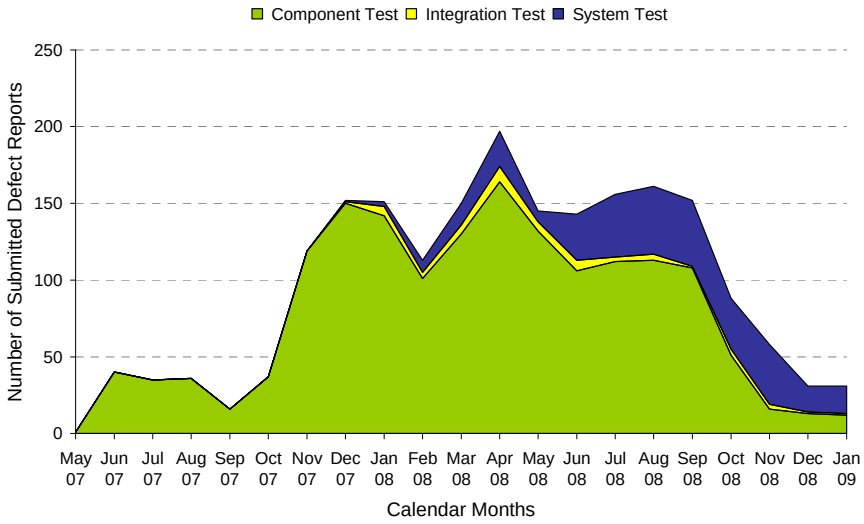


Figure 5. Monthly rate of submitted defect reports per test level

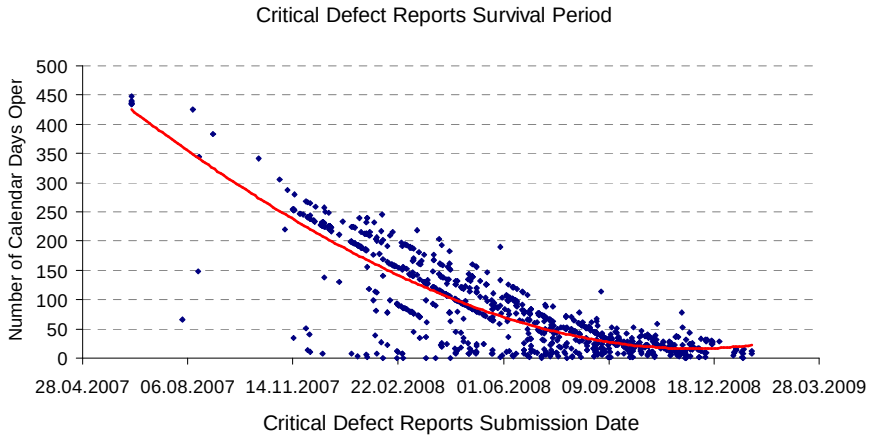


Figure 6. Survival period of critical defect reports

5. Conclusions

Looking at test execution progress, we see that most tests are executed within the beginning of testing a new release (see Figure 7). This met our expectations. Whether test execution goals can be reached or not can be therefore estimated early. If a testing period is subdivided into four weeks, as in our case, in the first week about a third of the test cases have to be executed, otherwise it is unlikely to reach the planned coverage. Reasons for not being able to execute about a third of the test cases within the first period could be that the application is still immature, the time-consuming test cases are executed first, or the capacity of testers is wrongly estimated.

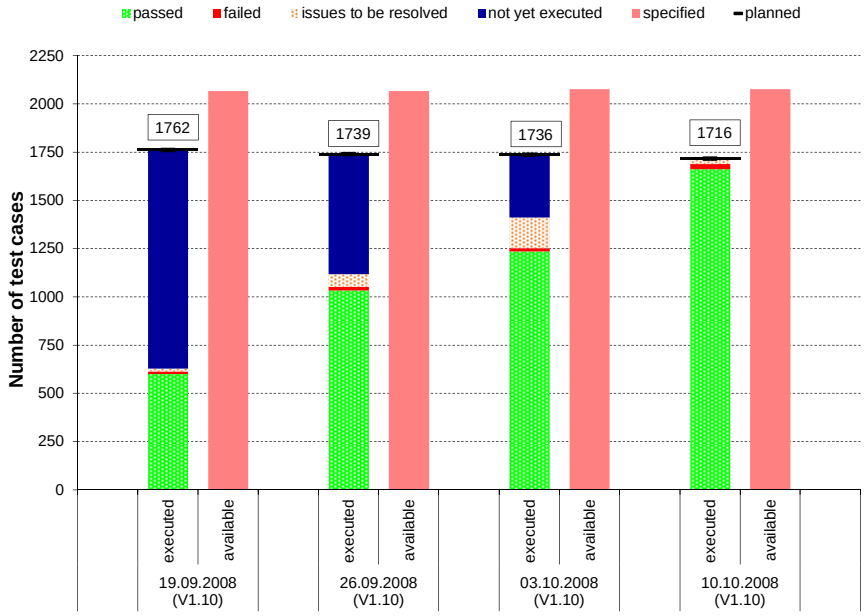


Figure 7. Weekly test-execution progress – improved presentation

The test case state “not completed” turned out to be misleading for management. The original idea was to allow the tester to express that he has started execute a test case but was not able to complete it due to some issues. These can turn out to be as a misunderstanding of the tester, a wrong test case specification, a change in the software specification not yet included in the test specification, or simply an indication of a defect whose circumstances need to be analysed more deeply. This state of uncertainty is difficult to convey to non-testers. We therefore suggest to replace it by a more comprehensive state, such as “issues to be resolved”.

Also a combined state, such as our described state “not run” is difficult to understand. It lead to many discussions in the project. Therefore, we would henceforth display Figure 1 (and likewise **Figure 2**) as shown in Figure 7. The “not run” state has been replaced by the unambiguous “not yet executed” and the number of specified (available) test cases is explicitly stated and does not have to be calculated by the viewer.

Overall, test coverage metrics were found to be very useful to evaluate the launch date at little cost to test management. While in early evaluation stages, the sheer number of unsuccessfully executed test cases and found defects was sufficient information, the test coverage information became more and more important during the course of the project. Our approach allowed to provide an overview, to qualify the findings, if necessary, and to focus on the exceptions by facilitating deliberate decisions by project management.

For the defect metrics, the major conclusion is: You can believe what is written in the books. In a real project the real data behaved as taught in the books, e.g. the ones listed in [1], [2]. The observation that follows is not mentioned in these books.

We used a five-value scale from critical, over major, medium, minor down to low. The strategic decision to process only critical ones before launch forced the defect board to investigate thoroughly whether a defect is critical.
As critical were classified defects that would prohibit production start, i.e. prevent the system to fulfil its purpose. Another reason would be the rare case where it does not prevent production but has a detrimental effect on testing, i.e. prevents testing or makes it disproportionally tedious. It could be worth to have a means to distinguish between production critical and test critical.

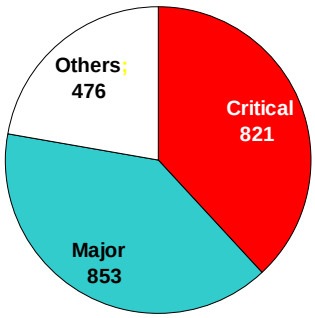


Figure 8. Severity of submitted defect reports

The fact that 2/5 of the defects were classified as critical and other 2/5 as major but only 1/5 of the defects ended up in the three less severe categories (others in Figure 8) indicates that a three-value scale would be sufficient. The subtlety of the five classes had no value for controlling the project and aggravates defect recording.

References

[1] Kan, S.H.: Metrics and Models in Software Quality Engineering. Addison Wesley, Reading, (1995)
[2] Amman, P., Offutt J.: Introduction to Software Testing. Cambridge University Press, Cambridge (2008)