Improving Software by Advanced Data Analysis

Rikard Edgren & Henrik Emilsson

EuroStar 2005, November 30

V1.0.2

Abstract - When developing software products a lot of data is produced regarding quality issues, e.g. test results on all levels, test coverage, bug reports, support incidents. This data is rarely used in the best way possible. There is more to learn about the data than what is received from simple reports delivered by the separate systems.

By looking for trends, patterns, outliers and unanticipated relationships in the different data sources, and by cross-referencing the data sources; a lot of new knowledge can be gained. This analysis will improve the product, and the way you produce it.

Multi-dimensional quantitative analysis, driven by the context, is an important complement to software metrics.

Rikard Edgren

Spotfire AB

Första Långgatan 26

S-41328 Göteborg

Sweden

rikard.edgren@thetesteye.com

Introduction

When developing software products a lot of data is produced regarding quality issues.

Software metrics can be extracted from these data sources to gain some knowledge, but it is not defined how and when they are best used [1]. The production of software is still a complex operation, therefore multi-dimensional analysis is recommended. By using several information sources in the analysis, new conclusions can be found.
The data analysis can only provide good answers in its context, with knowledge about the details.

The first step in the analysis process is organisation and access of data produced in software development. The second step is the more or less advanced data analysis. The third step is how you take advantage of the knowledge, and improve your process.

This paper will list a lot of ideas regarding these issues. You need to decide what is applicable in your context.

Access

The first questions are what data and analysis you already have in place, and which data you can make available. The data should be organised carefully in order to make advanced analysis possible. Cross-reference analysis is very powerful, but also sets higher demands on the organization of the data.

Important problems to address are: data in disparate formats, data is difficult to interpret, and difficult to log consistently.

Data requirements:

- *Consistency*
  If same data formats are used, comparisons are possible
  Use naming conventions to avoid misinterpretations
- *Identifiers*
  Using common fields with same identifiers enables cross-referencing between different data sources
- *Adherence to rules*
  If data is logged manually, typos and errors can disturb the analysis possibilities

The data should be useful in its context i.e., what is important for you?

To enable advanced analysis, also consider:

- *Bugs*
  Use dimensions like sub-area; bug type etc.
  Add keywords and other important information in reports.
  Orthogonal Defect Classification, (ODC) [2] enhances analysis possibilities.

- *Tests*
  Use dimensions like area, test type etc.
  Use mappings to requirements, bugs, support incidents.
- *Manual and automatic test results*
  Consider more information than just the result, e.g. environment, bug number etc.
- *Code coverage*
  Can be logged separately or not.
- *Source code repository*
  Change logs can be analysed.
- *Support incidents*
  Synchronise categories with tests and bugs.
- *Requirements*
  Use identifiers and synchronise categories with tests and bugs.
- *Project Plans*
  Log time spent in a consistent way and synchronise categories with tests and bugs.
- *Time Reports*
  Log time spent in a consistent way and synchronise categories with tests and bugs.

Adding categories to the data will increase the analysis possibilities. The more dimensions logged, the more analysis is possible, and also harder. So when categorizing a system, think about future analysis opportunities. It is also important to continuously maintain the classification system e.g., correct errors, add new items.

Data access tools might be needed for operations like joins and aggregations.

Analyse

A common problem when it comes to analysis is that it is hard to know which questions to ask, and that it takes a long time to get answers to the questions.

Simple metrics, like number of bugs, code coverage per function etc. is a good start; but the numbers don't tell much, and they might not be true. If it looks good, it isn't necessarily good. If it looks bad, it normally is.
An example would be 100% pass on unit tests that have statement coverage of 95%. When changing to multiple conditional coverage, there is only 30% coverage; and when system testing starts, a lot of bugs are found.

A good start is investigation known problematic areas and digging deeper into the metrics you already have in place.
Knowledge about details is necessary in order to find out what the answer is. By deepening the analysis, and asking more questions, more answers can be revealed. Just noticing that there are many or few bugs doesn't actually say so much.

Sometimes the answers might already be known, but now you have facts and figures to support your gut feeling.

Examples of questions during the analysis:

- *Bugs*
  Are there too many bugs; are they clustered in a specific area?
  Are there too few; but critical issues in a certain area?
  Can any trends be seen in bug appearance and fix rate?
  Has the percentage of UI-bugs gone down since we introduced UI guidelines?
  How does current project compare with previous?
  ODC - When were bugs introduced; and how long did it take for them to be found? Which development phase needs more attention?

- *Tests*
  Are there too many or too few tests?
  Do the tests cover all requirements?

- *Manual and automatic test results*
  Are the results what we expected?
  Has manual and/or automatic tests been well spent time?

- *Code coverage*
  What is the relationship between coverage and bugs?

- *Source code repository*
  Which "unchanged" areas have changed most?
  Where should we test more?

- *Support incidents*
  Which areas are sensitive?
  Are there a lot of general questions, indicating a weak help?
  Are bugs found in-house, but not addressed?

- *Requirements*
  How much time did we save by inspections?
  Do some requirements give more bugs in the resulting functionality? Why?

- *Project Plans*
  Can the bug system validate the plan?
  Which were the success factors, and how can it be repeated?

- *Time Reports*
  Are the results what we expect?
  Which activities are most well-spent time?

Visual analysis is much faster than just looking at numbers, and also more inclined to exploratory analysis, revealing answers you didn't know you were looking for. Ask many questions to explore the data, ask them quickly, and add more dimensions in the analysis to see more.

Trends and outliers are easy to spot in visual analysis tools like Spotfire DecisionSite [3]. Special statistical measures can also be used, e.g. for forecasting, or weighted measurements like quality metrics.

A bonus effect of the analysis is that you will learn a lot by analysing the data, both regarding the product and its process. The analysis can be a part of your daily work, and does not need to take too much time.

Act

The purpose of the analysis is to improve the product and the way it is produced.
If analysis reveals something during the project; improve current activities directly.
If post-project analysis is performed; improve process and planning. Post-project analysis is suitable for confirming that an approach paid off.

It is important to address issues in the right way. E.g. if no bugs are found; improve/change your test methods; and if many bugs are found; rather stop testing and instead improve implementation.

Knowledge about details is even more important when deciding how to act based on the analysis results.

Examples of actions and conclusions:

- *Improvements in bug reporting*
  During analysis of bugs you find that it is hard to analyse and therefore need to improve the way they are reported or the content.
- *Identify and meet goals for bug rates*
  Successful products share characteristics, find out which they are and use them.
- *Change test methods*
  Support incidents might indicate that the internal test method used is not catching the problems customer really have.
- *Switch test areas*
  Spot areas where testing would be more efficient and know when to stop testing parts of products.
- *Improve unit tests*
  All unit tests are successful but still there are a lot of bugs opened or manual tests are failing. This can be pinpointed down to a certain code block.
- *Better code reviews*
  All unit tests are successful but still there are a lot of active bugs and manual tests are failing.
- *Spot areas that will increase customer satisfaction*
  Compare support system with bug system. There might be not-so-severe bugs that would increase customer satisfaction.

- *Improve how requirements are written*
  Areas with a lot of bugs can be compared with the requirements written for that area. This might be due to poorly written requirements or that the amount of requirements is too low.
- *Better timing in project plans*
  Learn from past projects and use this wisdom in current project plan.
- *Support for project management*
  Use the knowledge learned and include in daily dashboards and status reports.
- *Decision support for release of product*
  Know when the quality is good enough for each sub part of the product.
- *Discover areas for training*
  If testers do not find bugs they might need training; if developers produce code with a lot of errors they might need training.
- *Increase test efficiency*
  Develop better tests and use another test methodology.

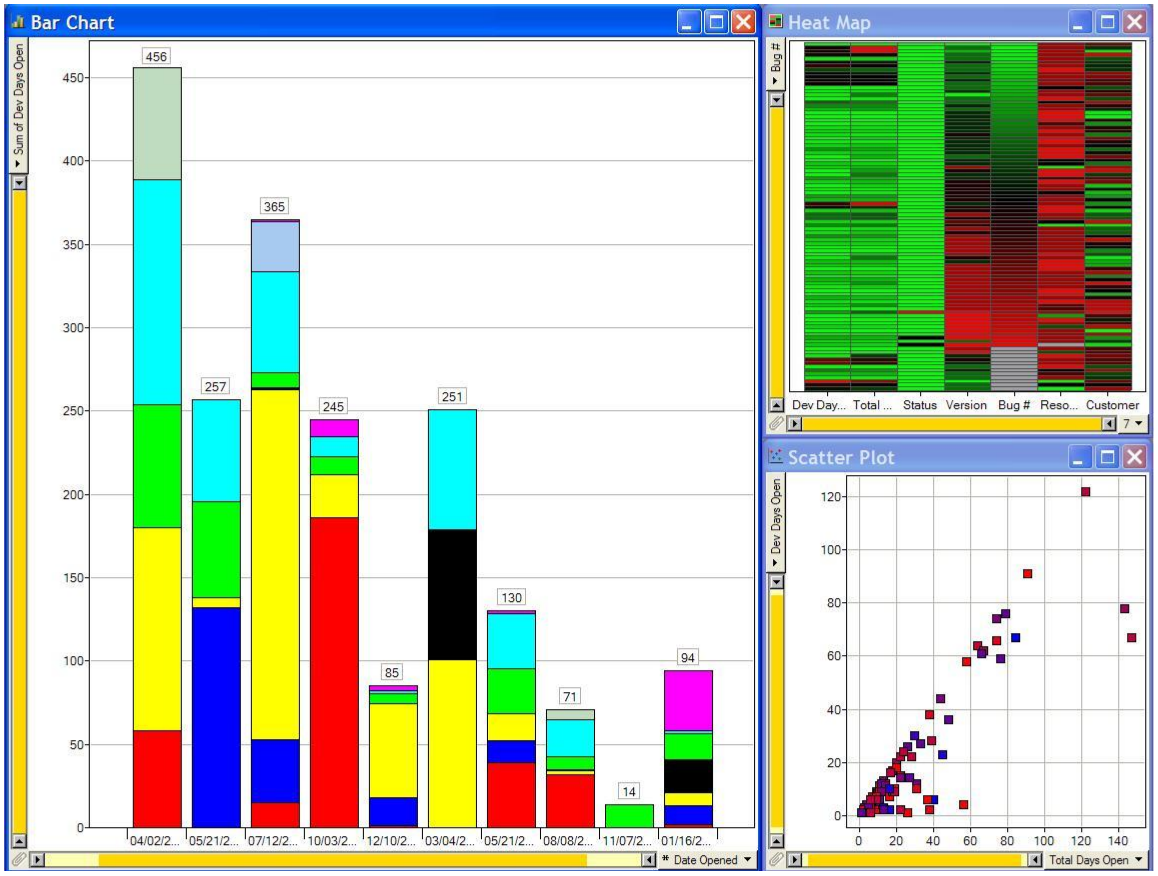
Defining actions that you want to do is actually a good way of starting your analysis. This way you need to think what questions you must ask to get an answer, and also what you need to do when it comes to organising the data sources.

The analysis results should be reported in appropriate manner. Having actual figures is essential to accentuate improvement suggestions.

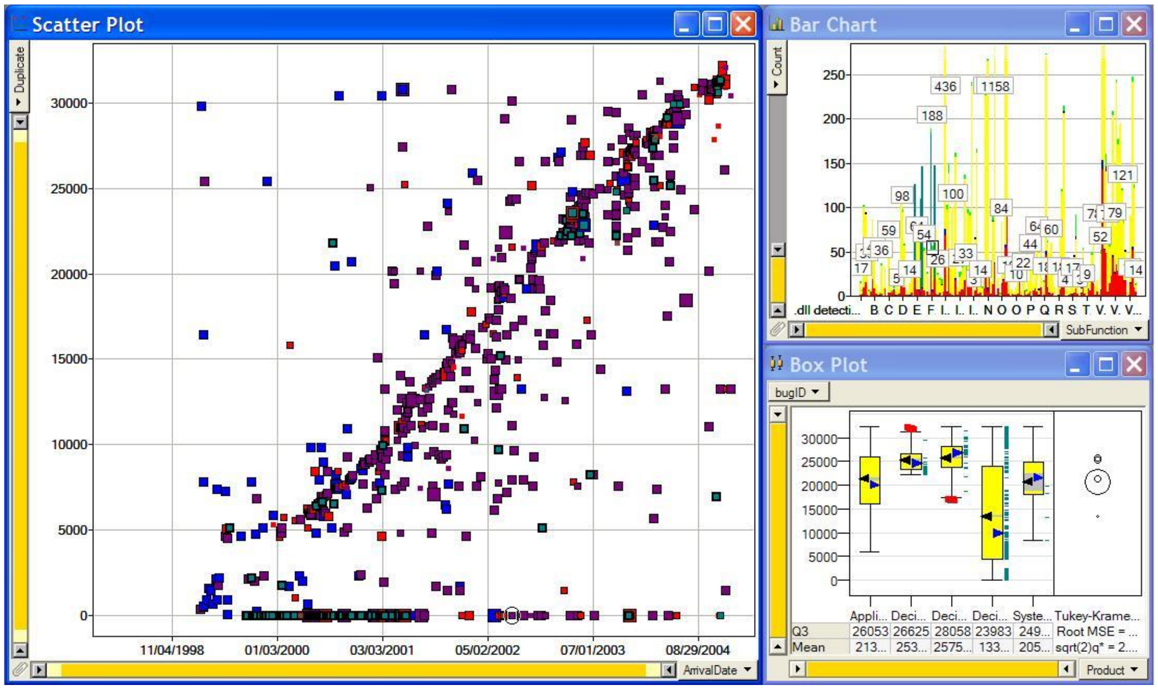
During or after analysis you will probably discover ways to improve data organisation; and ways to improve the data you have in your dashboards.

Finally; don’t forget to evaluate the analysis effort.

Examples



Support incidents per product coloured by status and time to address issues.



Few or many bugs can be both good and bad. Clustered bug shows weakness in one area; but at the same time it raises the question if test coverage really is good for the other parts.

References

[1] Cem Kaner & Walter P. Bond, *Software engineering metrics: What do they measure and how do we know?* http://www.kaner.com/pdfs/metrics2004.pdf *10th International Software Metrics Symposium (Metrics 2004)*, Chicago, IL, September 14-16, 2004.

[2] IBM Research; Orthogonal Defect Classification http://researchweb.watson.ibm.com/softeng/ODC/ODC.HTM

[3] Spotfire DecisionSite, http://www.spotfire.com