2

June, 2008

testing experience

The Magazine for Professional Testers

Test Management & Requirements
Experience & Tools

printed in Germany · free exemplar · www.testingexperience.com · ISSN 1866-5705

Treasuries from the dump - The software tester as an IT-archaeologist

by Dr. Konrad Schlude

Figure 1 - Wooden cups from the High Middle Ages, Stein am Rhein (Switzerland)[2]

Abstract

What has archaeology to do with software testing? From a simplified point of view, archaeology is about finding, securing, and interpreting human artefacts; software testing is about deriving test cases and running these test cases in a test environment. Whereas one of these topics is quite often "digging in the dirt" in not such nice weather and environment, the other is done in front of a computer in an air-conditioned office. The aim of this article is to show that the skills of a software tester can be used to do some kind of IT archaeology. Similar to an archaeologist who examines former dump sites and finds archaeological treasures there, we analyzed two areas of "dump data" of an IT system. With the results of this analysis, the underlying problems could be solved at the roots instead of fighting the symptoms. From the darkness of unnoticed garbage true treasures could be salvaged.

1. Introduction

It is possible to find treasures in the dump. Sometimes, archaeologists find very interesting things in former dump sites or mediaeval latrines. Especially in the deoxygenated climate of latrines, wooden objects can outlive several centuries. In their excavations, archaeologists have found mediaeval wooden cutlery and glasses frames. Outside this special climate, these things would not have survived the centuries.

So, a dump can give important information about life in former times [2].

1.1. Analysis of "Dump data"

Are there dark dump sites with hidden treasures in IT as well? Is there "dump data" that can be analyzed with profit? And what has testing to do with it?

In the following, we report on an IT project in which there was some kind of "dump data". Both in the development and production of an application data was accumulated and nobody took notice of the accumulation of that data. From our experience, this is the rule and not an exception.

2. The application and the development process

In order to better understand, we will give some details of the application and the development process. It is an application in the banking area. The customer company consists of several business units, and originally the application was developed for the prime business unit of the customer company. In the meantime, the application has become multi-tenancy, i.e., the other business units can work with adopted versions of the application.

The application can be used to choose a strategy from a list of proposed strategies, and the proposed strategies depend on the business unit (see Figure 2). Every night, batch processing checks whether the selected strategies are consistent with the current situation.

Besides an internal intranet version, there is an external version that can be accessed via the Internet.

It is a browser-based J2EE application with a mainframe component (CORBA, DB2, and batch) and a UNIX component (Oracle DB and batch). Therefore, there are several developer teams that are quite independent.

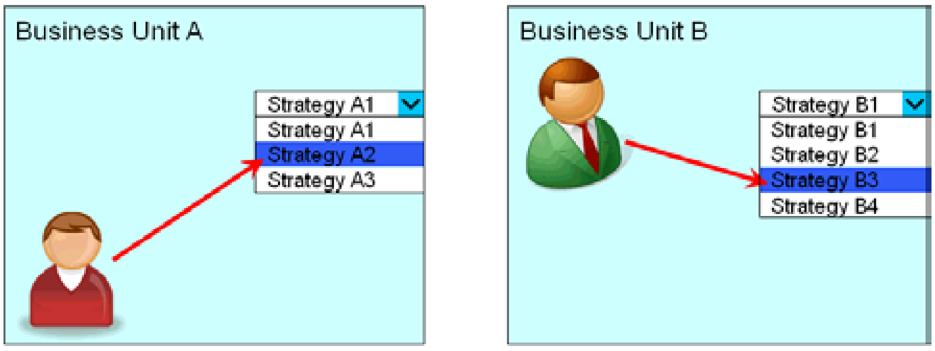

Figure 2 - Relationship between business units, strategies and clients

The application has been developed over a period of about five years, and there is further development. In the first phase, a lot of functionality was added. After that, the amount of new functionality was decreased, and therefore many team members left the project. Other external reasons led to "the great escape" from the project, especially in the requirements engineering team and in the JAVA team, many members left.

3. Error messages from batch processing

As mentioned above, batch processing checks the consistency of the selected strategies and the current situation. Shortly after this functionality had been deployed, the batch processing started to write error messages to a file. It was known that some of these error messages were related to external data delivery. Therefore, and since there was a lot of new functionality to develop, the error messages were ignored. Within two years, the number of daily error messages grew to 6,000. It became obvious that this number had to be reduced.

3.1. Results of the analysis

3.1.1. Changing the business unit

Most of the error messages (about 90%) were caused by changing the business unit. If a client is serviced by another business unit than before, this leads to an inconsistent situation (see Figure 3).

An automatic selection of a new strategy is not possible. In order to reduce the number of error messages, the application was enhanced to identify such cases on its own.

3.1.2. The -1 bug: quantum mechanics in IT?

A special problem was the so called -1 bug. In the database, there was one record with the strategy number -1 that stands for 'undefined', and this value -1 was not a legal value. Although there was only one such record, hidden in the large number of other problems, the problem was analyzed. This was due to the fact that the unknown reason could lead to other more serious problems. To analyze the situation, the developers were interviewed. The batch developers said that their program does not modify the value, the GUI developers said that there is a mechanism that blocks such illegal values. So, how could this value end up in the data base?

The situation set us thinking of quantum mechanics: a particle passes through a barrier although it has not sufficient energy for the passage (due to classical mechanics). Since the lead of the developers was a physicist, we called the -1 value a "quantum tunnelling'.

It took a long time to find out a way to reproduce the -1 value. It was a GUI bug, and in order to reproduce the -1, abstruse navigation through the application with more than 20 clicks was needed.

The bug was fixed by an improved consistency check. From a testing point of view, it was a very interesting bug as well. Quite often, developers respond to bug logs by claiming "But no user clicks this way!". The quantum tunnelling of the -1 bug is an illustrative example that users do everything they can do, although developers do not expect that.

3.1.3. Batch bug

For a special type of clients, the value of the business unit was updated in a wrong way and this led to similar error messages such as changing the business unit. In the beginning of the analysis, this problem was overseen in the large number of other error messages. This was one example of the well-known fact that a large number of uninteresting messages hide the interesting one.

3.1.4. Other bugs

Beside the problems mentioned above, several GUI bugs were detected just by clicking through the application. Especially the attempt to reproduce the -1 bug showed a way to generate inconsistent situations and exceptions.

4. Analysis of bug reports

Every year, there are several releases for maintenance and further development of the application. In each of these releases, several hundred bugs are reported and for defect tracking a tool is used. It is documented in which component the bug appeared, and for the JAVA part, the sub-component is logged as well. The number of bugs depends on the amount of new functionality. In one release with a lot of new functionality there were about 1,200 bug reports, in a maintenance release there were about 200 bug reports.

Before, there was no attempt to find a structure in the bug reports.

Important notice: Quite often, bugs are the results of cascades of misunderstandings. For instance, a stakeholder gives unclear explanations, the requirement engineer does not recognize this and writes unclear specifications, the developer does not recognize this and writes wrong code. In such situations, there is no "guilty guy" and therefore, it is the wrong attempt to evaluate the work of a developer by the analysis of bug reports. The fact that the analysis of bug reports should be done with regard to social relationship is a well known fact [1]: "It should be mentioned here that some questions can be very dangerous for the personal relationships with in the group and should, therefore, be addressed only in a very sensitive and objective fashion."

First, the bug reports were sorted by component. Most of the bugs were related to the JAVA component. Only a few bugs were related to the mainframe or UNIX, and there was no visible pattern. Therefore, the analysis was restricted to the bugs related to the JAVA component. Sorting by sub-component did not show any pattern, the bugs ranged from typing errors to serious exceptions. Note that this is an indication that the present division into sub-component is not helpful.

Another pattern became visible. Many bugs were neither regression bugs nor bugs of the new functionality; they showed up at the interface between a previous application and the new functionality. One example was the display of security numbers. In a previous release, it had been defined that the security numbers should be displayed due to the location of a user. However, the new functionality always showed the Swiss security number instead of ISIN or the German WKN.

This pattern could be found in other areas as well. For instance, the new functionality worked well for the internal version of the application, however there was an exception in the external version. For the various business units, similar effects appeared. The application worked well for the prime business unit; however, the implementation of the special requirements of the other business units was not correct.

27% of the bug reports were of this type.

Obviously, "the great escape from the project" had caused a loss of know-how. Both stakeholder and requirement engineers were focussed on the prime business unit, and therefore, the special requirements of the other business units were not specified well enough. The bugs were found in testing, since the test team was not affected by "the great escape ".

It is not surprising that the loss of know-how leads to bugs, however, the 27% showed that there had to be an improvement.

To reduce the number of such bugs, a checklist was provided. The most important "dimensions" were illustrated with examples. Especially, the special requirements of internal/external versions and the several business units are mentioned. The check list does not give answers; with the help of this check list, every requirements engineer, every developer, and every tester can ask the right questions to identify gaps.

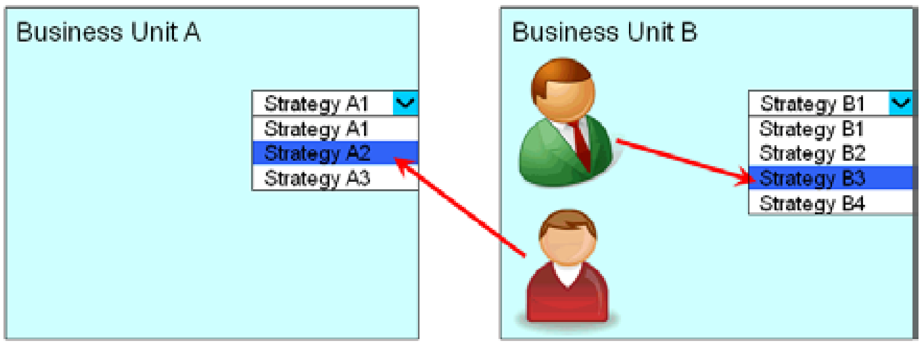

Figure 3 - Inconsistent situation after changing the business unit

5. Conclusions

Our conclusion is a contradiction to the notion of dump data: there is no such dump data. In principle, everything can be interesting and used to identify problems or to increase quality. In this article, we considered error messages from the batch processing and bug reports from the test process. In both cases, the sources of the problems could be identified and concrete suggestions for improvement were developed.
Most of the error messages were caused by external problems. However, several bugs were found as well. Whereas bugs in batch processing could be identified quite easily, the GUI bugs were hard to identify since there is large "distance" between the GUI and the error message of the batch processing. This is the reason why such an analysis cannot be done by a single developer. Typically, batch developers do not know the GUI, and GUI developers do not know the batch processing.
One reason for the importance of productive error messages is that bugs with low appearance priority appear in the mass test of the users. It is not surprising that the -1 bug was not found during testing, because no tester clicks that way.
Testing comes at the end of the development process. The result of testing, i.e., bug reports, can be used to gain information on the larger development process. This is similar to the error messages of the batch processing.
It is not a new idea that bug reports should be analyzed [1]; the well-known book [3] "How to break software" is the result of a bug analysis. Interestingly, analysis of bug reports is not part of testing education yet.
Bug reports and error messages are two possible "heart rate monitors" of an application. It is important to have such heart rate monitors. Another important aspect is to check these monitors.

5.1. Connection to software testing

There is one question to be answered: What is the connection to software testing? The answer is that for the analysis, typical tester skills are needed. One has to understand how the application and the development process work, the overview is needed. Furthermore, some "bug hunting passion" and the strong will to gain insight and to understand the problems are necessary.
From my point of view, the examples in this article illustrate how interesting software testing can be.

6. Bibliography

1. A. Endres:
An Analysis of Errors and Their Causes in System Programs
IEEE Trans. Software Engineering, Vol. SE-1, No. 2, June 1975, 140-149
Kantonsarchäologie Schaffhausen (Hrsg.):
2. EX TERRA LUX: Geschichten aus dem Boden. Schaffhauser Archäologie des Mittelalters
Schaffhausen, 2002
3. J. Whittaker:
How to Break Software. A Practical Guide to Testing
Addison Wesley, 2002

7. Wikipedia Links

- Batch processing: http://en.wikipedia.org/wiki/Batch_processing
- Multitenancy: http://en.wikipedia.org/wiki/Multitenancy
- Quantum tunnelling: http://en.wikipedia.org/wiki/Quantum_tunnelling

Biography

Konrad Schlude holds the Diploma in Mathematics from the University of Freiburg and a PhD in Theoretical Computer Science from ETH Zürich.
Since 2004, he is with COMIT AG (Zürich) and works as a consultant in the area of software testing. He focuses on banking software.
Konrad is interested in test automation and has presented some results of his work at the 2nd "SAQ Software-Tester Forum" in Zürich in 2006.

Among his interests in mathematics and software testing, Konrad is an active member of the town council of Jestetten (Germany).

[2] Source: Kantonsarchäologie Schaffhausen, Switzerland [2]