# Tales About a Small Software Testing Bridge from Academy to SMEs

Drs. Mark de Gids
Instituto Technológico de Informática
Universidad Politecnica de Valencia
Camino de Vera s/n
46022 Valencia, Spain
+34 963877069
mdegids@iti.upv.es

Dr. Tanja Vos
Instituto Technológico de Informática
Universidad Politecnica de Valencia
Camino de Vera s/n
46022 Valencia, Spain
+34 963877069
tanja@iti.upv.es

## ABSTRACT

This report describes our experiences during a University/Industry Interaction of type "technology transfer" and "consultancy service to help the company solve a, by them signaled, problem". We will describe the settings of the project, the progress of the project, the failures, successes and lessons learned.

## Categories and Subject Descriptors

D.2.2 **[Design Tools and Techniques]** *Object-oriented design methods.*
D.2.3 [**Coding Tools and Techniques**]: *Object-oriented programming, Structured programming.*
D.2.5 [**Testing and Debugging**]: *Testing tools (e.g., data generators, coverage testing), Tracing.*

## General Terms

Design, Languages, Theory, Verification.

## Keywords

Software testing, Small and Medium Sized companies (SMEs), consultancy services, academy-industry gap.

## 1. INTRODUCTION

This report describes our experiences during a University/Industry Interaction of type "technology transfer" and "consultancy service to help the company solve a, by them signaled, problem". In the next subsections we will describe the setting of the interaction.

### 1.1 The ITI

ITI, the Instituto Tecnológico de Informática, is a non-profit technological research institute located at University premises whose principal objective is to apply knowledge obtained from high-tech innovative scientific research to ICT-related small and medium size enterprises (SME).

The institute comprises 7 research groups, most of which are lead by University professors. The experiences described in this report are from our SQuaC group that is working in areas of Software Quality and Correctness.

ITI's mission statement says that one of the main goals is to build bridges between the knowledge and technologies learned from scientific research on the one hand, and the necessity for practical, simple and bullet proof solutions for the associated ICT related industries on the other hand. To achieve this goal, ITI helps companies with company specific education and courses, technology transfer, in-house developed off-the-shelf solutions, and in-company held consultancy projects.

### 1.2 The project

This article describes the process and side effects of one of these in-company held consultancy projects, where the gap between university and industry was more visible then ever before.

### 1.3 The company

The company involved is a small software house in Valencia consisting of:

- five full time contracted programmers (who are also responsible for the support and after sales services),
- a secretary, and
- a director, who is also for a substantial part of his time involved in code generation, besides that he acts as the software architect and project manager and is running his one-man sized marketing and sales department as well.

Their single and only product consists of an ERP-related software solution programmed in JAVA with some compatible hardware devices and they are financially doing very well due to some relatively big customers.

## 2. THE PROBLEM

It was the director of the company himself that came to us because he signaled a problem with respect to the quality, maintainability and/or scalability of their product. The software, initially programmed by just one or two developers, had grown from something small, transparent and easy to maintain, to something huge, opaque and impossible to maintain by his expanded group of developers.

In one of the first meetings, the director told us things like: "We know that we have some old source code incorporated that we really don't use anymore, but removing it from the system makes the program unstable or even sometimes not executable" or "Our customers are our test department, we don't test at all" or "We spend so incredible much time in debugging".

The director said that he thought that his problems could be solved if only he had an easy tool for version control, some way to be improve the source code traceability and that he wanted a "best practices" manual concerning these traceability issues. Moreover, conscious about the fact that they do no testing at all, he said that he wanted his programmers to know something more about how to test their own code and include this knowledge in the "best practices" manual..

## 3. SOLUTIONS

### 3.1 The wrong solution to the wrong problem

After these initial meetings, we erroneously thought we understood their problem and started to work on a "best practices" manual for "incorporating open source version and tracing tools" in their development process. However, due to the always existing and too short deadlines in real industrial environments and therefore lack a of available time by the programmers of the company to start applying the advice given in the manual, our "best practices" manual was never opened and until today it remains unread.

To teach the programmers more about how to test their own code, we gave them an off-the-shelf course on Unit Testing using the "JUnit" framework [1,3], based on the latest discovered techniques and strategies. The idea behind this was that once all the code would have been covered by unit tests, the program would contain less bugs, be easier to maintain and trace. However, we never finished the course because for the programmers it seemed to be "much too far from their daily tasks and realistic practical situations". They said they found it too theoretical and that they could not apply it to their code. The director said something like "this is always the problem with asking the Universities help, they do not know how things work in practice"…….

### 3.2 The REAL problem

Since unit testing and the JUnit framework are being used in many companies it are definitely not something purely academic, we figured that we needed to find out why the programmers thought it was not possible to apply these techniques to their programs. In order to be able to do this we needed to convince the director to share some source code with us to investigate why the content of the course was so far from reality to them. Although, many companies are not willing to share their source code with external parties, we finally convinced him and received a bunch of code that had a lot to do with a pan full of spaghetti: huge classes with and huge nested classes, huge un-parameterized methods with nested classes, cyclic calls between methods, no separation between the user interface, business logic and database persistence, no documentation, a lot of not understandable abbreviations and unreadable code.

The most surprising fact was that, although they claim to program object oriented and from a syntactical point of view indeed they do, in reality their code was not object oriented at all and therefore did not benefit from all the nice conceptual possibilities OO programming provides us.

It soon became clear that a tracing tool would not help them to unfold the spaghetti. Moreover, it became evident why they said that their code could not be unit tested, with these huge nested classes and un-parameterized methods it indeed seemed impossible to apply the JUnit framework (let alone design test cases that test the behavior of these huge methods).

How could a source code like this exist in a commercial environment in a company that is doing financially so well? Did the programmers of the company, all with a Master degree in Computer Science, forgot the design patterns they were taught in university, and what about the lessons on conceptual organization of software, not to speak about object oriented thinking, designing and programming. (We are not surprised anymore about the fact that most of the companies do not do any testing activity, nor know how they should do this.)

### 3.3 The REAL solution

Our objective was now to show the company that it were not the "theoretical" techniques that, as they put it "were not applicable in practice", but that the problem lies elsewhere! The way we wanted to do this was restructure part of their code, test it using the techniques described in the course and explain it all again but now using *before-and-after* scenarios. Together with the company we started a process to restructure the code. The aim of this process was not to rewrite the whole program, since that would go far behind the available time left for this project, but to redesign, rewrite and unit test an important component of the program. Subsequently, based on the before-and-after scenarios, we could then also write the desired "best practices manual" on software engineering and teach the programmers in conceptual organization of source code, object oriented programming, unit testing and database unit testing.

After studying the involved source code, we first made a list of requirements. Based on these requirements we wrote a first set of test cases for the test-driven development. We divided the code into a client, a server and a library with reusable code as can be seen in figure 1a and figure 1b.

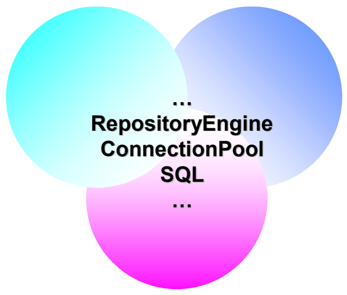

**Figure 1a. Before situation without code subdivisions**

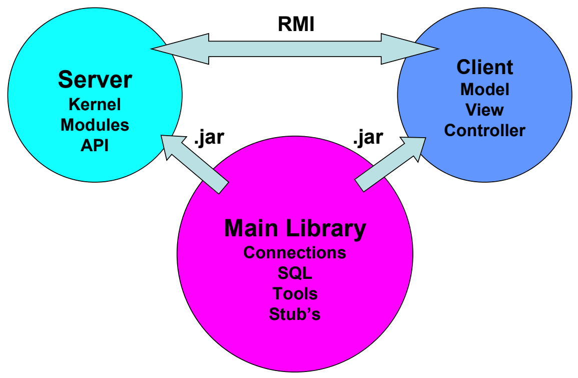

**Figure 1b. After situation with code subdivisions**

The client program was written using the Model View Controller (MVC) paradigm (e.g. [6]), and the server program was divided into an API layer, a kernel and a layer with modules around that kernel. The kernel was divided into units and these units into small classes. This process is visualized in figure 2a and figure 2b.

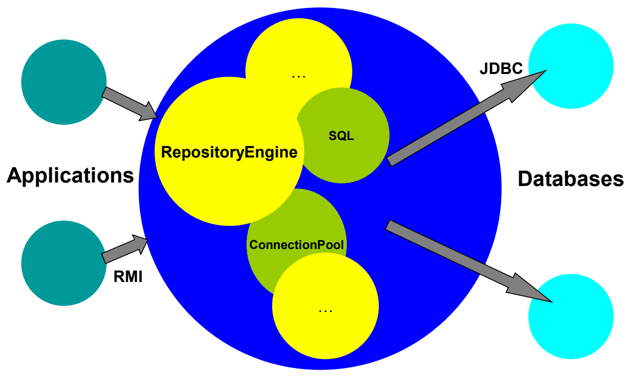

**Figure 2a. Before situation without conceptual separation**

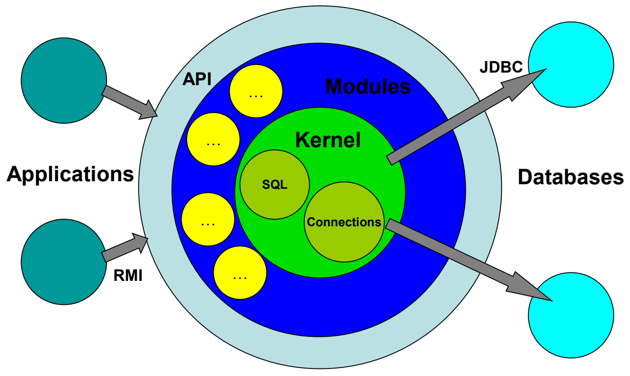

**Figure 2b. After situation with conceptual separation**

Every class was tested with a 100% coverage using normal JUnit Unit Tests [1,3]. The coverage was measured using EMMA [5].

The kernel unit with the database persistence code was unit tested using MockObjects [2] and DBUnit [4] integrations for JUnit. Figure 3a and 3b show how the whole program was tested.

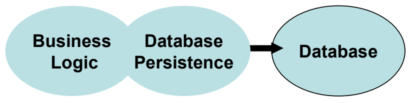

**Figure 3a. Before situation without layered database access and unit testing**

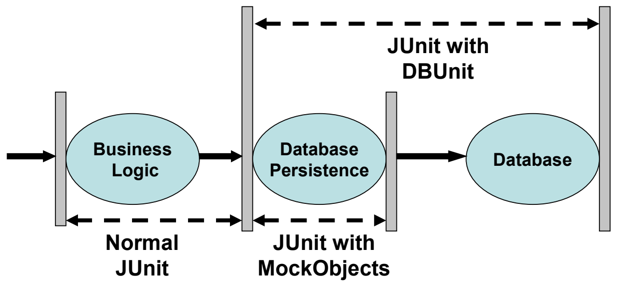

**Figure 3b. After situation with layered database access and unit testing and unit testing**

Finally we introduced the use of "interface" and "abstract" classes, and the use of "public", "protected" and "private" modifiers for methods and member variables. All source code was well organized in packages and well documented.

We held an in-company presentation showing the new restructured source code and comparing it to what they had. We showed them how easy it was to apply the unit testing techniques (that they had marked as "too theoretical") to this new code and why it was not possible to apply it to their code. We gave them a list of items they should take into account when programming and that was going to be used as the basis for the "best practices" manual. Although we were afraid and prepared for rejection and protest during our presentation, surprisingly the involved employees were really interested this time. They indicated that, being based on their own source code, made the examples and techniques a lot more clearer and directly applicable. Even the director was interested in the new source code organization and in the benefits of Unit Testing, especially in database unit testing. A new world opened to them and we are pretty confident that, when we evaluate the project within some months, the company is actually applying some of the techniques and advices that we gave them!

## 4. Conclusions

Concluding this report we would like to summarize the successes and failures, the lessons learned and what was gained in this project in order to help other related projects to succeed as much as we finally did.

### 4.1 Failures / Obstacles / Success

#### *4.1.1 Failures*

First of all the whole project took too much time due to the fact that in the startup phase we thought too fast that we understood the problem and sought the solution in a complete wrong direction. Secondly, due to the amount of time lost in wrong solutions, at the end of the project there was no time left for a good and appropriate follow up.

### *4.1.2 Obstacles*

The communication between a small company and a non profit scientific university institute can be difficult due to the lack of understanding each others jargon, priorities and communication habits. The project significantly improved when this obstacle was successfully over won.

### *4.1.3 Success*

Finally we succeed in building a small bridge between our institute and the company. The company is going to use the proposed solution but, more important: they got enthusiast in learning and applying the software engineering techniques we presented to them. Using their own code as a base and staying as near as possible to their programming habits, we succeed in demonstrating what problem they had and how it could be solved, in order to achieve a better product.

## 4.2 Lessons learned

First of all we learned that we should not immediately believe a company when they say that they know what they need. No matter how successful or good the company is, it might be that they know they have a problem but they do not really know what there problem is let alone how to solve it.

We learned that offering off-the-shelf courses on software engineering doesn't work for small and medium sized companies; the theory taught in such courses seems to be too far from the practice and daily activities of the people involved. In other words: is doesn't apply to their "reality". Using very concrete examples based on their well known own source code to explain a certain technique helps them to get a better overview and understand this new technique almost directly.

We also learnt that the skill level of the software engineers (all with Master degree in Computer Science) working for these kind of small companies is not always high enough to assure a good product. We where quite surprised when we saw the spaghetti style source code for the first time and about the fact that there was no testing department or any testing strategy at all. Understanding the syntax of a programming language is not enough to understand about software engineering concepts and benefit from the Object Oriented programming paradigm. Not knowing, and therefore not using, well known and well tested design patterns results in unnecessary reinventing the wheel, with all the risks of making small but crucial mistakes.

We learned that a solution running well in a controlled laboratorial environment does not automatically runs successfully in a commercial environment. In fact it took us quite some hours to make the restructured code work outside the lab and in their company.

Finally we learnt that a cocktail of open source tools (in our case JUnit, MockObjects, DBUnit, EMMA) are very successful to complete a certain task. However, solving the puzzle of combining open source tools to accomplish a certain task is not trivial and most of the time companies do not want to spend too much time on it. In our institute however, making tasty cocktails of open source tools is more and more becoming an independent research area with which we hope to be able to help many companies.

## 4.3 What was gained?

In later projects we found out that the case described in this report was just a tip of the iceberg and therefore we decided to change our strategy concerning this kind of projects. Instead of immediately offering off-the-shelf solutions and tools we now try to focus more on understanding the specific problems of the company and give them a tailored solution. Our services are more and more focusing now on the practical application of well know but too theoretical software engineering concepts.

But the big gain of this project is the construction and existence of a solid bridge now between the involved company and our institute. We overcame their skeptic attitude towards our scientific background by showing them good solutions based on their own code. We are delight to see their enthusiasm on quality improvement in software engineering and that the new code is going to be used in a commercial environment.

## 5. REFERENCES

[1] JUnit online documentation, http://www.junit.org

[2] MockObjects online documentation, http://www.mockobjects.com

[3] Massol, V. and Husted, T. *JUnit in Action*, Manning Publications, 2003.

[4] DBUnit online documentation, http://dbunit.sourceforge.net/

[5] EMMA online documentation, http://emma.sourceforge.net

[6] Gamma, E., Helm, R., Johnson, R., Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series, 1994.