Test and Quality Assurance in an Offshore Outsourcing Environment

Ronny Roos

A master thesis report written in collaboration with:

Department of Computer Science at Faculty of Engineering, Lund University - Sweden

and

Indpro AB, Bangalore - India

Advisor: Per Runeson

September 12, 2008
Abstract

In today’s software engineering projects, quality assurance and testing are common concepts. Often, firm procedures regarding these concepts are implemented in the project management. However, if these two concepts are combined with what is becoming common practice in many software development companies, namely offshoring and outsourcing, a more complex situation arises.

The focus of this thesis is the investigation on how to raise the code quality and how to assure that the software actually has a higher quality, through testing, in an offshored outsourcing environment. The work conducted at Indpro consisted mainly of the investigation to find the existing flaws in how the software was developed and how the work was conducted.

The data for the investigation was mainly collected through observations and interviews. To make sure that the focus of the data collection was correct, literature regarding similar areas of research was studied.

The second part of the work was constructing a solution to solve the problems found. The suggested solution consisted mainly of a new development process, which was delivered as a manual to the developers. Included in this was recommendations for test tools for different programming languages, a description of how to use the suggested tools and a description of how to create effective unit tests and include it in your development procedure. The manual also included basic recommendations regarding cultural differences and communication within the team and with customers. The final part of the solution was suggestions on how to reduce the effects of the offshored outsourcing environment. These suggestions were mainly meant for the management in Indpro to consider and introduce at later stages.

To make sure the suggested solution worked properly, the effects of the changes suggested were measured in a team and an analysis regarding the results was made. According to this analysis, the impact was positive. However, a longer study to confirm the positive effects of the changes has to be done to verify that the changes had a positive effect over a longer period of time.

To conclude, the impact of the changes had a positive effect on the quality of the
developed code during the time that the result analysis was done. The developers felt more safe with the new formalized process in place and have developed a more structured way of working. Hence, the conclusion is that the suggested changes fulfilled the goals set for this thesis, that is, raise the code quality, test the code to ensure quality and introduce better routines for the development teams.
The thesis was written in collaboration with the Department of Computer Science at the Faculty of Engineering at Lund University, Sweden and Indpro AB in Sweden and India. I would like to thank all the employees in Indpro for answering my questions, both developers and managers have been very helpful. I would also like to especially thank Tom Bergström and Erik Johansson from Indpro for input and suggestions to this thesis.

A special thanks goes to Victor Karlsson and Johanna Lynn who previewed the report and brought helpful suggestions on how to improve it.

Last, but definitely not least, I would like to thank my advisor from Lund University, Per Runeson, for much valuable input to ideas regarding both the focus of the thesis, the theory behind the investigations performed and the written report.
# Table of Contents

1 Introduction .................................................. 1

2 Background .................................................. 3
   2.1 Definition of Terms .................................. 3
   2.2 Indpro ............................................... 4
   2.3 Indpro’s Business Idea ....................... 4
   2.4 Programming Skills .................................. 4
   2.5 Cultural Differences ................................. 5
   2.6 Teams and Structure .................................. 6
   2.7 Quality Assurance Procedures .................. 8
   2.8 Test Procedures .................................. 8
   2.9 Programming Languages and Test Software .... 9

3 Problem Description ........................................ 11
   3.1 Problem Statement .................................. 11

4 Action Research ............................................. 13
   4.1 Situation Analysis .................................. 13
   4.2 Problem Solution .................................. 15
   4.3 Solution Implementation ............................. 15
   4.4 Outcome Evaluation .................................. 15

5 Results .................................................... 17
   5.1 Situation Analysis .................................. 17
   5.2 Problem Solution .................................. 20
   5.3 Solution Implementation ............................. 29
   5.4 Outcome Evaluation .................................. 29

6 Discussion ................................................ 35
## Conclusions and Future Work

### 7.1 Indpro specific conclusions

### 7.2 General conclusions

### 7.3 Future Recommendations

### 7.4 Outsourcing Environment

### 7.5 Development Process

## Appendix

### A.1 Development Process

### A.2 Interview Protocols

## Program Listing

### B.1 FibonacciGenerator.cs

### B.2 Automatically Generated Tests by .TEST
Offshoring and outsourcing have become more and more common in Europe, USA and the more developed parts of the world. The reason is that the resources (developers, engineers, software architects, etc.) have become very expensive. A software developer often has a minimum of three years of university training. This of course means that he or she wants a relatively high salary. In turn, this means that companies that develop software have huge costs for employing the engineers that develop their products. However, this changed when countries with lower wages, such as China and India, entered the software market. Several companies were of course interested in software engineers that worked at those prices. This was how the offshore outsourcing concept got started. With offshore outsourcing however, a lot of problems arose, such as cultural differences and difficulties in communication. This is a part of what this thesis covers.

The thesis mainly takes test and quality assurance into consideration, which is known in software engineering throughout the world. In most parts of the world this concept is a very important part of the daily work and software developers are fully integrated with the processes assure quality. However, this is not the case everywhere. Especially not in companies located in the parts of the world that have had less time to develop an advanced technical environment. This study focuses on one country in particular, namely India.

The focus of this study is quality assurance of code developed in offshore development teams in Indpro’s main office in India. The first area of this report is to present the environment as it was before changes were implemented. The next chapters are an investigation and a presentation of the results of the studies conducted. Last, is the discussion of the results and stating suggestions for further actions and changes.
Introduction
The circumstances for this thesis are somewhat different from other research regarding quality assurance. This thesis focuses on quality assurance in an offshore outsourcing environment in a very young company. This, in turn, leads to a development environment that does not have many of the features of similar environments in more technically advanced environments. Hence, the basic concepts of a development environment are not fully in place in the projects.

This chapter contains a description of the conditions in which most of the development teams produce their code.

2.1 Definition of Terms

Offshore Development
Offshore development means relocation of the development (all development or just smaller parts) to another country. This thesis only covers offshoring of software development but any kind of production can be offshored.

Outsourcing
Outsourcing is basically a subcontracting process, often conducted to lower costs in the outsourcing firm. The firm conducting the outsourcing hires another firm to do parts of what they previously did themselves. The term outsourcing is often combined with offshoring (offshore outsourcing) meaning that you are outsourcing to another country.

Distributed Development
Distributing your development means that you have several places for handling development, basically splitting up tasks between teams or persons (or other resources). For example: One team that takes care of all the design of the software and one team that takes care of all the actual coding of the design and the final product. These teams do not
have to be in the same office, in fact, they can be located in different parts of the world, that is, distribution of production within or across companies.

2.2 Indpro

Indpro is a small Swedish company with offices both in Sweden and in India. They have approximately 50 employees in both offices combined. In total they have about ten simultaneous projects running. The company started in 2005 and is still fighting to take market shares from the bigger companies in the same area of expertise.

2.3 Indpro’s Business Idea

Indpro’s business idea is to provide offshore development teams, not complete outsourcing solutions.

For example, a Swedish company wants to migrate their already existing application to C#.NET. They hire Indpro to do this and contract them for three programmers for a specified amount of time. If Indpro have the resources for this in house (available in the company), they simply assign the programmers to that customer. If they do not have the programmers available in house, they recruit them.

The Swedish company hires Indian programmers, that is, programmers to make an Indian programming team. Indpro simply helps them to find the programmers and provides the necessary equipment and administration personnel in India.

What they provide is not outsourcing solutions but more or less programmers for hire in an offshore environment with project management included.

2.4 Programming Skills

In general, the knowledge about the actual programming languages and understanding of how to develop certain items and/or parts of programs is high among the programmers at Indpro.

However, what is lacking in regards to programming skills is the overview a more advanced programmer has. He or she is working toward a greater goal, which is satisfying the customer and constructing a system that is as good and effective as possible.

In general, the programmers in Indpro are satisfied when he or she has gotten things to work in a proper way. Some of the programmers are, of course, just as aware of quality in the code as any advanced programmer.

Many programmers lack the basic understanding of how the programming languages work on the lower levels (how the code translates to machine code or similar levels), and hence do not know how to write effective code. The reason for this is not that they are bad programmers, their education simply does not focus on these things.
One should know that it is, of course, not all programmers that are this way. What is described is simply the general programmer in Indpro. What you hire, when you contract Indpro, is a competent programmer, not an engineer with his/her mind set to develop the best software possible to satisfy the customer. You do not get a software architect, you do not get a mathematical genius, you simply hire a programmer.

What one should also be aware of in this situation is that the general opinion of being a programmer in India is much different from what it is in Sweden. In India, a programmer is someone that earns much money and gets much respect. Hence, the occupation has high status in the society. This means that a lot of those studying to become programmers do not have the general interest in computers and/or programming that many students from, for example Sweden, do. Similar to this is the general computer knowledge in the country, most of the people in Sweden that study to become programmers or similar do so because they have a general interest in that area, people grow up using computers and develop interest in them. Most Indian programmers however never used a computer before they started studying programming in college. This leads to a huge difference in general knowledge in the computer domain.

The above statement is not based on any empirical studies; it is strictly observations and interview results from the research conducted here. For more information regarding the methods that have been used to analyse and collect data see 4.1.

2.5 Cultural Differences

Of course there are cultural differences between Sweden and India. The main difference, according to the observations, is the general mindset of a programmer. Programmers from Indpro, tend to:

- Not have the same attitude toward deadlines.
- Be bad at estimating tasks compared to the general programmer.
- Claim that they have understood even though they have not.

The above statements are observations done during the research in Indpro and were confirmed as actual problems in similar situations by Höfner and Mani [8].

These differences often pose problems in the communication between the Swedish customers and the Indian programming team. For example, if a deadline regarding delivery of a product is set on Friday, the product is not delivered to the customer, no mails/phone calls or any other notice has reached the customer. The customer, with right, often get very frustrated by this. In India however, this is nothing strange, time is most often not important. So, what happens is that the customer, now already irritated by the late delivery, gets his code. The code compiles, and works as it is specified in the functional
Background

requirements. However, the quality of the code is not very good. Some of the basic things like button placement in a graphical user interface may be handled very badly.

The programmers follow the specification, and if it is not specified where the button shall be placed, the button can end up being placed in a very bad place (from a cognitive perspective). This is not wrong, but it is not correct either. The customer probably expects the user interface to follow certain graphical guidelines based on recognition and other cognitive guidelines. That is, things that the general Indian programmer does not know about. So, why does the Indian programmer not know about this? Because many of these things do not matter to them. It does not matter if a newly constructed sidewalk is wobbly, as long as it serves it purpose, which in programming would be similar to thinking: What does it matter if this code is properly formatted or not? As long as it does what it is supposed to do.

These differences, of course, go both ways. Meaning that the Indian programmer can get offended by a very straightforward question from a Swedish customer, just as the customer gets irritated with the developer taking too much time to ask a simple question.

Mentioned above are just the most pronounced cultural differences. There are of course many more that are not listed since they do not matter as much as the above. These cultural differences have been found partly with the help of interview results (see A.2) and partly through observations done in Indpro (more about methods for collecting data in 4.1).

2.6 Teams and Structure

The teams in Indpro vary from one to nine persons in size. The teams have one customer each. The customer supplies requirements for the project and in some cases they manage their own team (providing team leaders, project managers and ways of operating).

However, in general, the teams operate according to Indpro’s definition of SCRUM (more about this in 2.6.1). They are provided with a Swedish (on site in Bangalore) technical project leader, which can help the team with difficult tasks in the project and also has an overview of how the work progresses. The project leader sits in on the meetings with the customer to help explain and straighten out certain communication issues which may arise. The project leader is also responsible to make sure that the team is fulfilling all the duties regarding SCRUM (entering data into the backlog, producing burndown charts etc.), the project leader is basically a SCRUM master (see Schwaber [16]) with some extended duties.

On special occasions (tricky code, uncertain projects etc.) the technical team leaders might even act as a programmer in the team, to demonstrate coding standards, help to solve problems and similar matters.

Indpro also has one resource (senior programmer/software architect) that can be placed in different projects to get an extra resource or support in that team.
2.6.1 Indpro SCRUM

The Indpro SCRUM is not very different from the standard version of SCRUM (for more information see Schwaber [16]). SCRUM, in general, is an agile way to manage a team. All the customer’s requirements are entered into a "Product backlog", prioritized (by the customer) and estimated by the development team (time estimation). The backlog items may be very big tasks which require breakdown to become manageable, this breakdown is done when the items are planned in the different "Sprints". The sprints are more or less a period of time where the team works on specified tasks (the tasks are picked according to the customer’s prioritization).

The progress of the work is recorded by updating the estimated time left on specific tasks by the end of each day. With the help of these updates, the team can create "Burndown charts" in which you can graphically trace the progress of the work. Here you can also trace anything that disturbs the sprint’s progress. You can see an example burndown chart with notes at the points where something happened to change the sprint in figure 2.1.

![Sprint 6 Burndown Chart](image)

Figure 2.1: Burndown chart

The Indpro SCRUM still contains the standard items such as SCRUM meetings within the team every morning to determine what each member is going to work on, and if something might be obstructing their work. The customer is also involved during some of the meetings in the procedure, not in the daily SCRUM, but most often in the "Sprint planning meeting", where it is decided what is to be done in the sprint. For more information regarding these SCRUM artifacts described and SCRUM in general, see Schwaber [16]. When running SCRUM, close contact with the customer is important to monitor his/her satisfac-
tion with the work so far and any change requests for future reference. In this environment, this is not possible. The customers are always hundreds of miles away, meaning that you can not have regular face to face meetings. Most of the meetings are done over Skype (Voice Over Internet Protocol program, used for conference calls over the Internet), which makes it important that what is said is clear and that nothing is misunderstood (since the participants can not rely on body language or similar to get further understanding).

2.7 Quality Assurance Procedures

In some of the projects, in Indpro, there are (by customer) specified quality assurance procedures. In the teams that there is a routine for quality assurance, it is most often flawed. There are no actual review or manual test procedures defined before release of code or final product to the customer (meaning that the code only gets tested manually by the developers according to his/her own test routines). This often evolves to irritated customers that have to perform bug correction themselves, or construct a new document containing all the bugs that have to be fixed.

This is very demanding and annoying for both parts, the developers have to fix bugs, and the customer has to spend his time reviewing code and producing bug lists.

2.8 Test Procedures

Some projects in Indpro have defined test routines that are defined by the customer. The ones that do not have this are most often only manually tested.

The projects with testing procedures often have good procedures defined, but they are not followed. For example, there may be a requirement stating that unit tests shall be done, and so they are, but how they are to be done is not stated in the requirement. Hence, it is up to the developer to write unit tests that he or she thinks is good. In some cases this works out fine, since the developer knows the weak points of the code he or she has written, but often he or she does not know what to test, and write unit tests that are useless.

To summarize the unit testing: Performed in some cases but far from an adequate amount of the classes are tested according to proper procedures.

Some projects use static testing, a form of testing that does not require the code to be run on a computer. It can be performed in the form of, for example, code reviews or programs that analyse the code and look for programming flaws. The projects in Indpro that use static testing mostly use forms of code review and in one case, a tool was used occasionally. However, there was no defined process for how to handle the results.

In summary: The projects most often run undefined test procedures, the ones that have defined procedures often lack certain important parts of testing in the procedure (such as what to test). The rest of the projects only use manual testing, as stated previously. This
does of course not mean that the code produced by these teams are of very low quality all together, what it means is that the quality is very dependent of the skill of the one who wrote it and what manual tests he or she performed on it.

2.9 Programming Languages and Test Software

Indpro handles a wide range of programming languages, everything from Delphi and PHP to C# and C++. This, in turn, means that a lot of different development environments are in place. The environments that are used often depends on what the client requests or simply what the developers like best. Different teams may also use different versions of the same development environment, to reduce problems with integration for the customer.

Indpro does not have licenses for any testing tools, and they do not have any personnel trained to specifically handle testing.
Problem Description

There are many articles and books involving testing and quality assurance. It is a rapidly growing area in which many advances are made every year.

What differs in this thesis is that the focus on quality assurance and testing, not to forget, in an offshore environment with limited resources.

So, what is the actual problem? - Indpro was simply delivering code with too low quality. The problem is: How do we raise the overall quality in the software?

3.1 Problem Statement

This thesis’ main focus is on quality assurance in software development to make sure software of greater quality is produced. This can be achieved through proper testing and proper routines according to authors such as Höfner and Mani [8] and Rompaey and Demeyer [15]. To achieve proper testing, first the programmers have to get the right attitude, that is, how to produce high quality code, through proper programming procedures.

In this context, quality means software with as few faults (bugs) as possible, that performs its work as effective as possible. Preferably the software is connected to a test framework which is integrated in the development environment to support effective regression testing.

This outlines the main problems to be considered in this thesis:

• How can the code quality be raised in the teams?

• How can testing be achieved in the projects while keeping to a strict budget?

• How does the offshored outsourcing environment affect the situation?

These questions bring the focus of the study and the handling of the problem to the following three main steps to do to achieve a solution to the stated problems.

• Identify steps of the development process that need to be changed and improved to achieve greater quality.
- Identify test tools that will give the most benefit to the largest amount of projects without requiring special training.

- Identify how the effects of the problems regarding communication and cultural differences in an offshored outsourcing environment can be reduced.
Chapter 4

Action Research

To solve the problem at hand both research and action were required. Hence the method defined as "Action research" in Höst et al. [10, 3.5] was used. First, there is a need to define what is wrong or what is missing in the situation. The second step is to find a solution to that problem, implement the suggested solution, extract results from the implemented solution and based on those results make a suggestion of what can be done to improve the situation in the future. Finally, the solution is evaluated. This chapter covers these steps in the method and describes all of them extensively. The description further explains how the steps defined in Höst et al. [10, 3.5] is applied to the problem area of this thesis.

4.1 Situation Analysis

During the situation analysis, an evaluation of the current flaws in the environment in Indpro was done. To get the most accurate result without being biased, the evaluation was done in several teams, using different languages and programming environments. The projects had different project managers and different customers but the developers worked in the same office using the same procedures. The flaws were written down during the analysis, later they were confirmed through different scientific articles regarding similar areas of research. Some of the articles and other research material (primarily Höfner and Mani [8], Munkebo Christiansen [6, 5] and Oza et al. [14]) also helped to guide the analysis and contained valuable suggestions to find new information.

4.1.1 Data Gathering Methods

For the situation analysis, data was gathered primarily in two ways, namely through:

- Observations
- Interviews

The observations were conducted in two different ways. In two projects they were conducted by an active participant, where the author’s role was to lead the team and analyze
the situation. In the rest of the projects data was gathered by an active observer where the teams were aware that an analysis was conducted. For more information regarding the roles of the observer, see Höst et al. [10, 6.3]. The observations were done during work hours in the teams. That is from 09:00 to 18:00, during which the teams operated normally. The primary part of the observation data was gathered during the first month of the study.

The observations were collected on a computer and later used to construct interview questions. The interviews can be found in A.2. The interviewees were selected from the senior staff in Indpro, one technical project leader, the process manager and two developers were interviewed. The interviews were semi structured, which means that the main questions were simply used as a guideline to keep the conversations on the subject.

The interviews were conducted at separate occasions and the two interviewees (described above) did not discuss the interviews with each other. The data collected during the interviews were then summarized and can be found in A.2.

4.1.2 Data Analysis Method

After the data was gathered it was analysed using a method of semi scientific nature. Since no method for classifying the collected data was found, it was analysed using an investigative method which is very dependent on the analyst (since it is based on the analyst’s own conclusions and thoughts). To get the method more scientific, phrase intensity was used, meaning that the analyst tried to keep a list with the most commonly described problems in Indpro. This data was collected both from interviews, observations and informal conversations regarding such issues. The list had no statistical or mathematical relevance, but was simply kept as notes to make the analysis more accurate. For more information regarding methods similar to this, see Höst et al. [10, 6.6].

The analysis of the data from the interviews was basically conducted through making a summary of the most important points (the interviews were several hours long in total and were hence not transcribed). The decision of what to exclude from the interviews when summarizing them, was made using partly the phrase intensity data and partly through reasoning from the interviewer (that is, the subjects that most often came up during discussions and interviews were considered more important and were hence summarized and added to the report). This data was then compared to that collected from the observations to assure higher accuracy. If answers from the interview and observations did not concur, the data from the observations were used. This in turn means that there is a high dependency on the observations, meaning that they have to be performed as objectively as possible to ensure a correct result from the analysis of the data.
4.2 Problem Solution

The problem solution was created with the input from the situation analysis and the input from the main problem areas specified by Indpro. This means that not all of the problems found in the situation analysis can be dealt with using the solution created since the goal of the solution is to remedy the main problems specified.

The problem was solved in different steps:

1. Define main problem areas
2. Find possible solutions for the problems
3. Document solution
4. Evaluate solution

4.3 Solution Implementation

To implement the suggestions in this thesis, an implementation plan was developed. According to the observations of how the teams operate, it would be beneficial if the process was introduced during bigger project milestones or if it was introduced in the startup phase in new projects.

To introduce the process to the teams the CTO (Chief Technology Officer) in India studied the process extensively to be able to answer questions. Similarly did the project leaders. The process was then introduced to the team members by the project leaders. The team members got a copy of the suggested development process A.1 and got time to study it.

The rest of the changes (see 5.2.5) suggested in the thesis were simply studied by the management which could then introduce them verbally at a good time in the projects.

When the actual development with the new process started, the project leaders supervised and made sure that everything in the new process was followed. After a few days (depending on how well the team handled the new process) the supervision lightened to give the developers more time to learn by themselves. After this stage in the implementation the process was considered to be in place in the team.

4.4 Outcome Evaluation

To evaluate the result of the implemented suggestions a case study on one of the teams was performed. Since there was no previous development or test process in place, comparing the new situation to the old became hard. To solve this problem, old estimations and burndown charts were used as data to measure any change in estimation accuracy.
Since there was no previous testing in this project, there were no old test cases to compare to. Instead interviews were designed to extract information from the developers regarding items that had no other data available for comparison. The interviews were structured and used only the questions available in A.2.2. The answers were finally analysed and the results of the solution evaluated.
5.1 Situation Analysis

This part of the result contains descriptions of the different problem areas that were found during the analysis. The problems are listed, described and connected to relevant literature.

5.1.1 Requirements Specification

Requirements are the heart of software development. If we do not know the exact requirements, we are likely to spend time on developing the wrong things. Meaning that we are charging the customer for things he/she does not want or does not need. So, one of the most important parts of successful software development is specifying and handling these requirements correctly. Currently, in Indpro, the customers specify all the requirements. No requirements specification and/or research regarding the requirements is done by Indpro.

The requirements come in a wide range of types. Some are specified as informal requests over Skype chats or meetings, others come as use cases and some even come in the form of a design suggestion with attached written requirements. This however does not mean that there are requirements specifications delivered which can be followed from beginning to end. The requirements continuously get updated during development.

This puts Indpro in a situation where they are very dependent on the competence of their customers. The customers (and Indpro) have to take all the difficulties about requirements into consideration when working with the requirements. Things that need to be considered are (among others):

- Imprecise requirements.
- Tacit requirements (unknown requirements).
- Lack of customer knowledge.
- Lack of customer involvement.
Munkebo Christiansen [5] mentions all of these as possible problem sources for specifying incorrect requirements in offshore environments. Since Indpro is running a large variety of project, most of these problems are present in some way. What causes the most problems seem to be lack of customer involvement and unknown requirements. The teams often get requirements stating that a type of functionality shall be developed. What is not stated is usually how the GUI should handle this. The customer often leaves this part out of the requirements. The problem is that the customers often expect that the GUIs are developed to fit good design principles. What they do not think about is that most of the developers in Indpro have not had this kind of training, meaning that the lack of specification for these kinds of requirements often lead to major failure.

Some of the customers also regard project offshoring as a way to not have to care about the project any more (once they have written the contract they just want to get the final product without further involvement from their side). This is however a very bad way to handle offshored projects (or any other project). Even if you have paid to have a project done offshore, you (as a customer) still have to supply the same guidance as with regular projects. That is, you still have to have regular communication with the team members, give regular feedback regarding the progress and so forth.

According to Munkebo Christiansen [5], these flaws in the process can account for a major part of the errors conducted in the projects present.

5.1.2 Test Environment

To ensure quality, a proper testing environment is recommended. This will of course raise the development time compared to what it is without developing tests. But according to Rompaey and Demeyer [15], an effective test suite can make up for this in later stages of the project.

As mentioned briefly earlier, there is no test environment set up at Indpro. Definitions for what standard tools to use are lacking, and most often testing is not done at all. Again, according to Rompaey and Demeyer [15], even if it is only unit testing done properly this will raise the quality.

Since there is no defined test environment, there is no personnel with specific knowledge in testing (no test teams), according to Faiz et al. [7] this will affect the quality of the code negatively.

5.1.3 Communication

The communication within a development team is very important. Indpro has the advantage of having most of their teams in the same place, meaning that they can use direct (synchronous) communication. This is an advantage compared to other teams in similar situations that have to rely on email, chat or similar methods of communication according to Munkebo Christiansen [6].
But, as mentioned earlier, the customer and developers, are still using email, chats and other non face-to-face communication methods, meaning that much information can be lost.

Mistakes in communication can always happen, even though you use synchronous communication and have regular face-to-face meetings. According to Siakas and Balstrup [17], one of the more important things to do, to prevent this from happening is to make sure that the different cultures can work smoothly together. This is partly solved in Indpro through the Swedish project managers on site in Bangalore, which can act as a middle man and get everyone involved to understand each other. But as pointed out, it is only partly solved since the project managers are not present all the time. And since the project managers are there, no effort is spent on developing new routines to get the cultures to work smoother together.

Munkebo Christiansen [6], brings up the issues regarding communication between clients and development teams (with different languages). This problem is partly solved in Indpro through the Swedish project managers. The project managers participate in almost all planning meetings with the customer, and can often solve many language or cultural related issues. However, the project managers can not always be there, for example during e-mail communication between customer and developer. This in turn can lead to misunderstandings between both customer and developer, as well as between project manager and developer/customer.

5.1.4 Customer Relationship

Indpro’s customer relationships varies a lot. Ranging from close contact customer relationships with meetings over Skype on a daily basis to projects where the customer and developer speak to each other every other week. The wide range of different relations between customer and developers also poses problems to the organisation itself according to Oza et al. [14] and Nguyen et al. [13]. The problems have not yet grown to be serious problems in the current organisation. Since the customers themselves often decide the amount of communication they want to have with their team, Indpro can not take full responsibility for the communication issues.

5.1.5 Software Development Process

With no software development process in place, quality of the code often plummeted in teams with programmers that did not have a very structured way of handling their development. According to Jarvenpaa and Mao [11], a formalized process to handle development is much preferable to no process at all when it comes to raising the code quality in the teams. To put it in the words of the actual authors:

"In the software development literature more broadly, many have found that the increased levels of formalized routines in systems development improve quality
Other research, such as that conducted by Braun [4], points out the importance of a good control procedure for the projects. In some points, this exists in Indpro, but in others it is totally lacking. For example, project managers might sometimes get notified too late of changes in their project, or might not be notified at all due to unofficial meetings. The article mentioned ([4]), regards larger companies, and larger teams. But since it is already a problem in the small teams in Indpro, the reasoning in Braun’s article should apply here too.

5.1.6 Analysis Summary

According to Höfner and Mani [8], the three cornerstones in offshore development are communication, coordination and control. All of these have been discussed in the analysis. What is important, is that all of these things need to work together to be able to create an effective offshore model. If one of them fails, the model will not completely fail but it will certainly be flawed.

According to the analysis of the situation, there are multiple areas that need to be worked on to create a better development environment.

5.2 Problem Solution

This section contains possible solutions for the defined problem areas. The main problem areas are then connected to form the outline of a solution suggestion, which is described in detail.

5.2.1 Possible Solutions

The primary focus of this thesis is defined in 3.1. Following are solution suggestions for the separate problems and finally the final solution suggestion.

Raising Code Quality

Follow the principles of Beck [1] and make sure that all the teams operate exactly according to the defined SCRUM methodology by Schwaber [16]. By doing this we have established a way of developing code that is exact and strict. According to Jarvenpaa and Mao [11], firm routines should raise the code quality, if they are followed. The problem however is that different projects have different rules and requirements and have to be managed differently. Because of that, applying all these strict principles in all projects is not a good idea. To be able to implement all of these principles, an extensive training would also be needed.
Hence the suggested solution for this problem is a lighter set of principles that the developers themselves can study on their own, since proper tutors for more advanced principles are not available.

**High Quality Testing With a Strict Budget**

To completely solve this problem, you would need to develop code tested by experienced testers. The code should for example follow the guidelines presented by Rompaey and Demeyer [15], which discusses the composition of well constructed and effective unit test suites. Since we are working with a strict budget in this problem, hiring trained test personnel with expert knowledge is not an option.

A solution to the problem would be to train the existing personnel in creating effective unit test suites using methods similar to those of Test Driven Development (see among others Beck [1]). Additional training in how to construct effective test suites would also be required to make this solution effective.

**Decrease the Effect of the Offshored Outsourcing Environment**

To solve this problem, the relationship between customer and company needs to be completely restructured and operate differently from now. Authors such as Höfner and Mani [8], Oza et al. [14] and Nguyen et al. [13] have defined different frameworks and recommendations to use in environments similar to this. These frameworks are most often entire suggestions for how to manage the relationship between customer and the department or provider handling the outsourcing. With the help of these frameworks it is stated that both quality in the code and the relationships between customer and provider would be improved.

However, bringing in such extensive changes as the ones suggested by for example Höfner and Mani [8] was not an option. That would have required so many changes in team structure, management and added new requirements for the customers (since it would also had required them to change) that the budget would have been exceeded.

Hence, a solution requiring less restructuring was required. A possible solution is therefore to construct a smaller framework for Indpro, by using the suggestions taken from the articles like Höfner and Mani [8] and Nguyen et al. [13] and modifying them to better fit the situation in Indpro.

**5.2.2 Solution Suggestion**

According to much of the literature studied, a main concern when trying to raise software quality (especially in an offshore outsourcing environment) is to have a stable process in place (regarding both coding and how to handle communication and other issues related to the customer, see for example Höfner and Mani [8] and Nguyen et al. [13]). Meaning that this is the main focus when solving the problem.
To raise quality even further, there is a need to test the developed code, to make sure that the quality is high enough and that it fulfills the requirements.

According to the observations in Indpro, the main concern is to create a stable development process that the employees can follow. This development process also have to include the test environment and principles in how to conduct effective testing. The process also have to include items to help with communication and other offshore outsourcing issues that can arise.

The secondary focus of the solution suggestion is finding a test environment (for .NET as specified by Indpro) that fulfills the requirements set in the problem description. The complete solution suggestion is hence a combination of the suggested individual solutions.

The rest of the chapter is dedicated to describing this solution in detail. First to be described is the main part of the solution, namely the documented solution the development process suggested to use in Indpro. This is kept at a very basic level since it is the first formal development process Indpro will use and there is a lack of personnel able to train the developers in more advanced procedures. The general idea of this process was that it should be able to be taken into use by all the developers on their own. The development process also contains basic ways to handle communication within a project. The reason for this is to give the management an opportunity to start implementing other suggestions which are based on these basic concepts in the future.

Second a comparison and a conclusion for different test tools is listed. Finally the results regarding improvements in the development related to outsourcing are listed. The final suggestions here are intended for the management of Indpro and not for the developers. These suggestions are to be implemented step by step when there is an opportunity to do so.

To summarize, the solution contains two parts. The first part is the suggested development process to be used by the developers in Indpro. This contains basic concept to handle cultural differences and some communication issues that may arise. Included in this development process is also the suggested test environment, a description of how to use it within different projects and the basic concepts of what parts of a class that might be important to test. This part was written as a short manual for the developers to follow and can be found in A.1.

The second part is the one intended for the management in Indpro. This part contains suggestions on how to reduce some of the offshored outsourcing environment’s effects on the results of the team. These results are just found in this report and have not been printed in any manual. Compared to the development process, this part of the solution is very small. The reason for this is that the main focus was set on raising the code quality in the teams, according to the observations done and the analysis of the data collected this is best done through the change in the development process suggested.
5.2.3 Development Process

The suggestion for the development process can be found in A.1. This is accompanied by a checklist (also found in A.1) to use while developing. The attached suggestion is an actual copy that was introduced to the developers in Indpro.

The process describes everything from communication within the team to how to do effective unit testing on your code in a short manual. The general idea behind this is to make it possible for the developers to train themselves, using this simple manual. The steps described also contain short motivations to why they are done, so the reader can understand the importance of the steps.

A number of different books and articles were used to compose this suggestion and the rest of this section provides the information regarding where the suggestions in the attachment come from, and why they were used.

The main reason to create a new development process for Indpro was simply that they did not have any specific procedure in place. Much of the studied literature (for example Höfner and Mani [8] and Beck and Andres [3]), suggested well defined procedures while in situations similar to Indpro’s and hence, a process for both communication, team management and development was needed.

To assure that the code developed looks similar, general coding standards were introduced. The suggestions for coding standards are based on what is recommended from major suppliers (or producers) in the different languages (i.e. Microsoft, Sun Microsystems, etc.). The process also contains a chapter regarding Object Oriented Programming. This may seem unnecessary, but according to the analysis performed here, a reminder of the benefits of Object Oriented Programming is a good idea. The reasoning behind Object Oriented Programming is mostly fetched from Holm [9], Weiss [18] and Martin [12]. These books all bring up important parts of programming, everything from how to do basic programming to more advanced design patterns and good principles when developing software.

The chapter regarding communication in the development process was added due to, partly the fact that the observation and interviews conducted showed signs of communication issues and partly since many articles (such as Munkebo Christiansen [6] and Mani and Höfner [8]) stress the fact that communication is very important. Especially in a situation similar to this. The communication chapter in the development process will in no way solve all the problems. What it is intended, is to give a general guideline regarding what is expected from the developers in terms of customer contact and other communication issues. This has to be accompanied by practice and training of communication skills to be of any use.

The actual development process (coding, testing, etc.) was developed from the agile methods defined by primarily Beck (and Andres) in [3] and [1]. These articles together with the first chapter in Martin’s book [12] define a well structured agile development method. The development method was then modified to fit the SCRUM model in place at Indpro and then further modified to fit the different languages. One major element that was not
mentioned in the solution suggestion is the "Test first" mentality or so called "Test driven development" (for more information regarding this, see Beck [2]). The reason for this is that according to the observations and interviews conducted, the general programmer in Indpro is simply not ready to change their way of thinking that much. This does not mean that a programmer in Indpro can not learn how to use this. What it means is that with the current time frame and budget, there is not enough time or resources to train the programmers in this practice.

5.2.4 .NET Test Environment

Below are the results from the comparisons of different test execution tools and test environments for .NET. The tools that are compared are the following:

- .TEST by Parasoft (http://www.parasoft.com/)
- TestMatrix by Exactmagic (http://www.exactmagic.com/)
- FitNesse (http://fitnesse.org/)
- Combination of NUnit (http://www.nunit.org/), NCover (http://ncover.org/) and FxCop (Microsoft tool).

The tools are compared in different categories, such as general properties (table 5.1), cost (table 5.2) and test abilities of the tools (what kind of testing can be done, table 5.3). This comparison ends with a general recommendation of the tool that was considered overall best, with a description of why it was selected. The tools properties have all been evaluated by the author. To add further value to the evaluation, input from the team in which the changes were to be made were used (they had previously used similar tools and a discussion regarding these was held).
<table>
<thead>
<tr>
<th>Name</th>
<th>VS2005-integ.</th>
<th>VS2008-integ.</th>
<th>Languages</th>
<th>Ease of installation</th>
<th>Ease of use</th>
</tr>
</thead>
<tbody>
<tr>
<td>.TEST</td>
<td>Yes</td>
<td>No</td>
<td>.NET</td>
<td>Very easy</td>
<td>Very easy</td>
</tr>
<tr>
<td>TestMatrix</td>
<td>Yes</td>
<td>Yes</td>
<td>.NET</td>
<td>Very easy</td>
<td>Medium</td>
</tr>
<tr>
<td>FitNesse</td>
<td>No</td>
<td>No</td>
<td>C#/Java</td>
<td>Very easy</td>
<td>Hard</td>
</tr>
<tr>
<td>NUnit/NCover/FxCop</td>
<td>Individual integ.</td>
<td>Individual integ.</td>
<td>.NET</td>
<td>Very easy</td>
<td>Hard</td>
</tr>
</tbody>
</table>

Table 5.1: General properties of the test tools

<table>
<thead>
<tr>
<th>Name</th>
<th>Installation cost</th>
<th>Education cost</th>
<th>License cost</th>
<th>Usage cost*</th>
</tr>
</thead>
<tbody>
<tr>
<td>.TEST</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>TestMatrix</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>FitNesse</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>NUnit/NCover/FxCop</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

* The time spent when developing and running test suites - Usage cost.

Table 5.2: Cost properties (1-5. 1 = minimum, 5 = maximum)

<table>
<thead>
<tr>
<th>Name</th>
<th>Unit tests</th>
<th>Test coverage</th>
<th>Acceptance tests</th>
<th>Static code analysis</th>
<th>Code review support</th>
<th>Unit test generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>.TEST</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>TestMatrix</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>FitNesse</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>NUnit/NCover/FxCop</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 5.3: Test types supported
Tool Improvement of Cross Cultural Communication

The improvement of cross cultural communication can, according to Hőfner and Mani [8], for example be achieved by using the same vocabulary regarding project terms. Using test tools, this vocabulary can be extended to include, for example, the progress in testing, what specific methods fail, coverage of the code etc. Most importantly, you also get a number of new terms for the quality in the software, that is, the amount of tests written, test coverage, what tests fails and so forth.

The tools evaluated have of course got different properties and will help the communication in different ways. One of them is how they affect the words we use when we communicate, that is, the vocabulary used. The section below presents the results from the comparison regarding the tools’ effect on the vocabulary used when communicating:

**.TEST:** Adds a common vocabulary for both the team and customer in form of standard unit tests and coverage. It also makes communication between more experienced users and novice users more difficult since the experienced user is most probably using features in .TEST that a novice user does not know about. This situation can for example occur when an experienced developer tries to communicate the progress of testing to the customer (which is probably not an experienced .TEST user).

**TestMatrix:** Adds a common vocabulary based on the standard test and coverage tools for .NET. Since this tool is a module for integrating a test environment in Visual Studio less features exist than in for example .TEST. This will make communication between developer and customer simpler since both will be using features and language on the same level (the tool does not include any particular advanced features that an advanced developer can use). This in turn will cause communication to be more straightforward compared to, for example, .TEST.

**FitNesse:** Adds a common vocabulary for acceptance tests. The difference between the two previous tools and FitNesse is that FitNesse does not use the standard test language and methods and it is designed to perform acceptance tests. This in turn means that the vocabulary will differ from the tools that use unit tests. FitNesse is also a tool that requires an environment outside your normal development environment (the tests are written in a separate environment, namely the FitNesse server). This in turn means that the vocabulary and communication around the tool can become even more complex (since a separate environment is required and can be subject for discussion).

**NUnit/NCover/FxCop:** Adds the same vocabulary as TestMatrix with the difference that all the tools are external (not integrated in the development environment). Compared to TestMatrix it also adds the aspects of static testing to the vocabulary.

The vocabulary is of course not the only thing to regard when discussing communication. However, to keep the communication as simple and direct as possible, the vocabulary becomes important. It is important that everyone involved in a project can understand
each other. When you add the issue of cross cultural communication, you end up with an even more complex communication situation, where it becomes even more important to keep the communication simple.

To keep the vocabulary as simple, well known and as close to standards as possible, the best option (according to the comparison) would be to use TestMatrix or the combination of NUnit/NCover/FxCop. The tools themselves have different properties and abilities, but when it comes to the communication issue, they are very similar. TestMatrix has the disadvantage (when it comes to communication) of being composed of standard tools inside a module inserted in the development environment. This leads to a larger vocabulary (since you can discuss the module itself and its integration etc.).

NUnit/NCover/FxCop on the other hand, is not included in this way, but instead have to be run separately outside the development environment, which will have a similar effect as TestMatrix or Fitnesse (which is completely separate). In this package, static testing through FxCop is included leading to an expansion of the vocabulary since it adds more functionality.

To summarize the analysis of the tools’ effects of cross cultural communication; a common vocabulary is important when working in a team. It becomes even more important when the team operates using different languages. To simplify the communication within the team and to the customer of the team, it is, according to Höfner and Mani [8], important to use a common vocabulary. Therefore, to easier establish this common vocabulary, it is important that the vocabulary is as simple as possible, so everyone involved in the discussions can understand them easily.

The conclusion is hence that the usage of either TestMatrix or NUnit/NCover/FxCop is recommended from a communication perspective.

Tool Evaluation Summary

Based on the above comparisons .TEST is recommended to work with from a technical perspective, since it contains all the necessary features, and some extra features such as generation of standard test cases for written code. An example of this can be found in B.2 which is an automatically generated test for the class found in B.3.

The generated test cases actually (most of them) test proper aspects of the code. But they are hard to read and to understand for someone who is not used to reading them. This in turn leads to that most of the use with the automatically generated test cases disappears. If Indpro was a larger organisation with a bigger budget, .TEST would definitely be preferable. But for now, the benefits from .TEST would be far too small for Indpro (since they do not have trained testing personnel, and would have to train their personnel extensively to be able to handle a tool such as .TEST).

.TEST also has the disadvantage of adding a more complex vocabulary than the smaller test environments do.

This in turn lead to that TestMatrix was chosen as a final recommendation. It is a
small module, that is relatively cheap and integrates both in Visual Studio 2008 and Visual Studio 2005 and enables a simple unit testing environment with test coverage data inside Visual Studio. Since the module is small and does not require any external environment and uses the standard types of tests, it is also recommended from a communication perspective.

5.2.5 Offshore Outsourcing Environment Improvements

There are few suggestions in the previous sections that handle the specific effects of the problems caused by the offshored outsourcing environment. These problems are instead handled separately in this section.

The results presented here are those intended for later use by the management and were not implemented fully when evaluating the outcome in 5.4.

The main problems are listed below, and solution suggestions presented. What you should keep in mind when reading this is that only what can be put in place with the current environment and resources are listed as solution suggestions. Other items which can be realized when the company has more resources available are listed under 7.3.

Requirements Specifications

According to Munkebo Christiansen [5], teams in similar situations to those located in Indpro have benefitted from using short iterations when it comes to requirements (and development). To achieve that in Indpro, new requirements should be posted before each sprint planning. This is to ensure that the team stays agile and can follow the customer’s demand more accurately than if the requirements come in during the sprints.

Improving the situation regarding tacit requirements and lack of customer involvement can be achieved through better communication and customer relationships.

To improve imprecise requirements from the customer, assure that he or she has technical personnel with skills in requirements specification to write the requirements before signing a contract with him/her.

Communication and Customer Relationships

As Munkebo Christiansen (among others) mentions in his articles [5] and [6], it is crucial for teams to have good communication. Both with the customer and internally. The communication issues in Indpro are not yet major issues, but to solve the problems that do exist, the following changes are recommended:

- Developers must have more frequent communication with the customer.
- Synchronous communication between customer and team needs to be increased.
- Language and communication barriers need to be handled more effectively.
According to Munkebo Christiansen [5] many projects benefitted from having close communication with the customer. According to the same study, synchronous communication is preferrable. Keeping this in mind, a good solution would be:

Ensure that the team gets the right start through frequent (set a minimum amount of meetings per week) synchronous meetings over Skype (since that is what is commonly used in Indpro) with the customers. Before the meeting the team leader should make sure that all the developers are ready for the meeting and have prepared relevant material for the meeting (questions, task estimations, etc.). During these meetings, the relationship between customer and developer will evolve, which should decrease the general reluctance to initiate communication that the developers have in the initial stages of customer contact according to Höfner and Mani [8].

Höfner and Mani [8] also states, that what is important here is to educate the developers in the cultural differences that exist between Swedes and Indians. A direct question to, for example, a Swedish customer is often the quickest way to an answer, meaning that the developers have to learn to take this route to get quicker answers. The customers, of course, also have to adjust to the situation. They have to realise that it is not always proper to ask an Indian a direct question since it may seem rude to them. Hence a middleground has to be reached.

With this solution we have both ensured a closer connection to the customer as well as more effective communication between development team and customer. This will decrease some of the effects of the problems regarding the language and communication barrier.

5.3 Solution Implementation

The results of the implementation of the suggested solution are simply that the process was welcomed by the developers, team leaders and the CTO. The CTO’s role in the initial phases was smaller than expected (not many questions were directed to him), and the team leaders’ role grew larger (since they had to answer more of the questions).

The team members started using the process immediately. What was first implemented by the developers was the recommended commenting standards. What took the most time to implement was the test routines and the iterative development.

5.4 Outcome Evaluation

This section contains the details from the evaluation of the effects of the implemented solution. This evaluation was done as a case study (see Höst et al. [10, 3.3]) on a team that used the suggested solution. Keep in mind that the solution that was implemented is the development process suggested in A.1 and the test environment.

To provide an accurate picture of the evaluation a background of the specific team and project is given. The code developed by the team itself (with most of the changes in place)
was then examined and compared to the code developed before the suggested changes were implemented (see table 5.4). A study on the accuracy of the estimation of programming tasks was also conducted to get input regarding the changes in the estimation accuracy.

To get further information regarding the impact of the changes, the developers in the team were interviewed three times, the summaries of the interviews can be found in A.2.2. The main purpose of these interviews was to get an impression of what the changes meant for those subjects to them and to extract information not available as code or other data from the project.

5.4.1 Team Background

The team consisted of two experienced developers, both with more than three years of experience in this domain.

They were running a .NET project (using Visual Studio 2005) to migrate a specific application from Delphi to .NET. The actual migration was already done when the suggested solution was implemented in the team. The solution was implemented when the team had reached a larger milestone in the project. This part of the project was going to be strictly development in .NET, using a third party control as a main component in the application. The task was to exchange an old third party module with a new third party module. This in turn means that much of the code was already written, and was simply going to be modified to fit the new application.

This is not an optimal environment to test the new process, since much of what is described in the solution suggestion is best to use when developing new code. Instead, it was agreed that all the code that was modified was going to be handled according to the solution.

Previously, no test cases were written, and no documentation standards were used. The team also had a hard time to make accurate estimations of the tasks.

The team had very little contact with the customer before the solution was implemented.

Previously the team worked with a non existing specification, their only task was to migrate the code to C#.NET. At the milestone at which the changes were implemented, the project requirements were defined by the team itself, instead of having just one big task in the backlog (see Schwaber [16] for more information regarding backlogs for SCRUM projects).

5.4.2 Outcome Analysis

Data to use in the evaluation was collected primarily by interviewing the two team members using the interview questions which can be found in the interview summaries in A.2.2. The results from the interviews were analysed according to the method in 4.1.2. The interviews were conducted on three separate occasions (one interview in each sprint during the study).
The interviews were conducted in the middle of the sprint and were conducted with both developers at the same time.

Furthermore, the code itself and the sprints’ estimations and burndown charts (see figure 2.1) were used as data to analyse the outcome. To analyse the code, comparisons between the code developed before the suggested changes were implemented and code developed with the changes in place were done. The same approach was used for the estimation accuracy, where time estimation data before and after the implementation was compared.

Developers View of the Outcome

Overall the items suggested for implementation in Indpro have been welcomed by the developers. To change the way the work is actually conducted took alot of work and was a slow process. What was most appreciated was the formalized way of working, stating how they should proceed with development and how to complete a specific task.

The developers also feel that the quality of the developed code has been raised. Partly with the help of the unit tests and partly with the help of the formalized process which they can follow when they perform a task. The developers also feel more assured that they are conducting their programming according to standards that are satisfactory for their customers while working in the suggested manner.

Changes in the Developed Code

To evaluate the changes in the developed code, classes and forms containing much of the logic in the system were compared. A comparison between how the code looked before the suggestions started to be implemented in the team, and after, was done. The code was evaluated twice, once during sprint 19 and once during sprint 20 (see table 5.4). No comparison was made in the first sprint due to the fact that virtually none of the suggested changes were implemented at that time.

<table>
<thead>
<tr>
<th></th>
<th>Pre implementation</th>
<th>Sprint 19</th>
<th>Sprint 20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Well structured code</td>
<td>Mostly</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Descriptive comments</td>
<td>Often</td>
<td>Very often</td>
<td>Very often</td>
</tr>
<tr>
<td>Documentational comments</td>
<td>Often</td>
<td>Few</td>
<td>Yes</td>
</tr>
<tr>
<td>Unit testing</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Test coverage</td>
<td>0%</td>
<td>0%</td>
<td>3%</td>
</tr>
<tr>
<td>Testing quality*</td>
<td>None</td>
<td>None</td>
<td>Medium</td>
</tr>
<tr>
<td>Explanation of algorithms</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 5.4: Properties in the code and how they were fulfilled.

* Quality of the tests refer to that the tests are actually testing things that are important for the application.
Changes in Time Estimation Accuracy

The next step in the data collection was comparing the time estimations (estimations done by the developers to estimate the time they would need to spend on each task) for the project. That is, comparing how much estimations differed from what was actually produced, before and after the suggested changes were implemented. The estimations in the tables 5.5 and 5.6 are measured in hours.

<table>
<thead>
<tr>
<th>Sprint</th>
<th>Initial estimation</th>
<th>Actually produced</th>
<th>Estimate difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sprint 6</td>
<td>104</td>
<td>52</td>
<td>-52</td>
</tr>
<tr>
<td>Sprint 7</td>
<td>126</td>
<td>40</td>
<td>-86</td>
</tr>
<tr>
<td>Sprint 8</td>
<td>144</td>
<td>84</td>
<td>-60</td>
</tr>
</tbody>
</table>

Table 5.5: Time estimations before implementation of solution suggestions.

<table>
<thead>
<tr>
<th>Sprint</th>
<th>Initial estimation</th>
<th>Actually produced</th>
<th>Estimate difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sprint 18</td>
<td>152</td>
<td>152</td>
<td>0</td>
</tr>
<tr>
<td>Sprint 19</td>
<td>160</td>
<td>184</td>
<td>24</td>
</tr>
<tr>
<td>Sprint 20</td>
<td>168</td>
<td>184</td>
<td>16</td>
</tr>
</tbody>
</table>

Table 5.6: Estimations after implementation of solution suggestions.

The first sprints were chosen in the middle of the development. Hence the project is not in a startup phase and not in an end phase, but more or less in the same state in all the compared sprints. This comparison merely shows how much the teams accuracy in their estimation in the sprint planning has changed with the new process. The sprints were selected randomly (with the rule that the sprints were not allowed to be either in the end stages or the startup stages of the project) and no data was compared before selecting what sprints to use in the comparison.

Analysis of Outcome Results

As you can see in tables 5.4, 5.5 and 5.6 above the overall quality of the team’s output have risen. The estimates are more accurate and the code is well structured and contains both documentational and descriptive comments in the code itself. One of the more important parts of the analysis is that the developers themselves feel that the quality of their developed code has risen. They also feel they now have a more structured way of handling tasks when developing.

This means that we have achieved what was specified. The quality has risen, the developers have a more structured way of working and most important, the developers are confident that they are writing code of higher quality than before, which the comparison showed that they actually did.

However, this project is, as stated previously, not optimal to evaluate the solution in. There was not much logic that was developed since most of that had already been done.
before the implementation of the changes. This in turn meant that the team could not implement many of the suggested ways to act when developing code (since their job now was to simply change a component). Instead they used the suggested changes on all the code that they came in touch with when exchanging the component (that is, they added comments, unit tests etc. for the parts of the code that were somehow related to the parts of the project they were currently working with). This means that the accuracy of the comparison lessens, since the team could focus solely on adding the suggested changes instead of having to both develop code and implement the suggested changes at the same time.

When it comes to the unit tests of the project, much of the code was heavily dependent on the user interface. Even more of the code was dependent on the actual third party component that was to be used. The reason for this was that the business logic was not separated from the user interface when the project was started. This caused much problems for the developers when they tried to develop unit tests. According to themselves (this is information from informal interviews), this is the reason for the low test coverage and the very late implementation of unit tests in the project. The reason that the unit tests’ quality has been rated ”medium” (table 5.4) is that the written tests often only test the most basic functions of the classes (like getters, setters and methods for comparison). The reason for this is, as stated previously, the more advanced functions are heavily dependent on the user interface and the third party component.

Analysis Summary

To summarize the effect of the introduced changes the categories described by Höst et al. [10, 6.2.4] was assessed. The categories are as follows:

1. **Effect**: How have the changes affected the situation? Was any quality improvement achieved?

2. **Cost of implementation**: How costly was it to introduce the changes?

3. **Usability**: How easy was the method to introduce? How well does the changes support the users’ tasks?

4. **Degree of automation**: How much can you automate the changes?

5. **Acceptance**: To what level have the users of the suggested changes accepted it?

According to the analysis done the outcome was as follows:

1. **Effect**: According to the analysis data and the observations done during the evaluation period, the effect on the team’s code quality has been positive. Meaning that we have fulfilled the goal of rising the code quality in the teams.
2. **Cost of Implementation:** The cost of introduction is very low since it consists purely of short internal training, a period of learning for the team members when they are studying the document and cost of coaching the team in the practices after implementation. After this it takes a while for the team to get all the suggestions in place. While in this stage the team will produce things a bit slower than usual. The expenses that are included when implementing (slower development speed, training, etc.) the changes suggested in this thesis are estimated to be covered by the time you earn by not having to correct as many bugs at later stages in the project.

3. **Usability:** The method is not very easy to introduce. It is heavily dependent on the individual’s ability to learn. Meaning that the developers themselves have a big responsibility in the implementation of the changes. When it comes to the suitability for their tasks, the changes are modified to fit the general projects in Indpro, so the support is quite good.

4. **Degree of automation:** It is not possible to automate the changes.

5. **Acceptance:** The users have accepted the changes well, but the implementation time (time it took to introduce all the suggested changes in a project) was longer than expected by the author when designing the solution.
The work in Indpro started out with three questions that needed to be answered. Namely:

- How can the code quality be raised in the teams?
- How can testing be achieved in the projects while keeping to a strict budget?
- How does the offshored outsourcing environment affect the situation?

It is important to understand that all of these three questions are related to each other. The quality issue is partly dependent on the testing environment as well as the effects of the offshored outsourcing environment. This means that all the results will have effects on the other.

Some of the discussion is tightly connected to the Outcome Analysis 5.4.2, this chapter brings up other issues to discuss than those discussed previously.

The first and main item that was solved was the code quality. The suggested solution for raising the code quality is the new development process which was put in place in one team in which the effect was evaluated. The problem is that the evaluation was done under the period of six weeks, before these six weeks, no development process was in place and the developers were not at all used to working in the suggested manner.

This in turn means that the developers are going to take time to first getting to know the new work procedures and even more time to modify their existing work procedure to follow the new procedure. Since the measuring of the effects were limited to six weeks, the accuracy of the data will be lower than if more time was given to the developers to learn the work process (which in turn means more time for data gathering and hence, higher accuracy).

The next problem was to find test tools to achieve a proper testing environment. This problem was limited to deal with .NET environment. The study focused on basic tools that did not require any specific training. Hence, many tools were not even considered and examined. Thus it can not be guaranteed that the solution is optimal. The suggested solution also misses an economic evaluation on a higher level. That is, what tools give the highest value compared to what it adds to the test environment.
The last problem to be solved was the offshored outsourcing environment’s effect on the entire situation. The solution for this problem is based solely on observations, interviews and the suggestions from scientific articles regarding similar situations and problems. To achieve better results and recommendations, this should have been accompanied by a deeper analysis of the exact effects of the outsourcing effects. However, this was not possible since all the projects in Indpro operate differently. Hence it would require a number of studies, which is not within the time limit for a master thesis. Instead a general study of the environment in Indpro was conducted and the results come from that general study.

What should also be kept in mind when considering the results of the analysis is that the developers who were interviewed regarding the development process might be biased since they might suspect that the interviewer wants positive answers to the questions asked. The solutions to the given problems in this thesis have also been compressed and mixed and are hard to evaluate all one by one, as stated in the beginning of this chapter. Hence the main evaluation of the solutions suggested was on the code quality of the teams and how they feel that they as individual programmers have been affected by the suggested changes.

Regarding the case study in the team (see results in 5.4), the results are inaccurate, since (as stated), the implementation was done in a team with bad conditions to handle the changes. Also, the time for the case study should have been extended to get a more accurate analysis. Preferably the solutions should have been tested in several teams with different conditions, this would have given a more accurate view of the effects of the implementation.

To summarize, the problems stated have all been solved. However, to get a complete solution or analysis of the situation, more time than is available for a master thesis would have to be spent on investigating the circumstances. This investigation would have to include extended research on the individual projects and their settings instead of generalizing the projects and working with general data collected.
Conclusions and Future Work

The conclusions for the work can be divided into two areas, general conclusions and conclusions that are specific for Indpro. That is, what conclusions can be made from the specific environment settings that Indpro had, and what conclusions are more general and could apply to a wider range of settings.

7.1 Indpro specific conclusions

With regards to the discussion in chapter 6, the conclusion is that the problems stated are solved but the solutions have some flaws. Most of the flaws come from the time issue of the project. The main goal was to assure quality and testing at Indpro. Using first and foremost the development process suggested in A.1 the quality of the developed code increased. A test environment will also be set in place, defining test tools to use and how to use them. To further increase the quality in the team there are suggested improvements in how to handle other aspects of the development, such as requirements, communications and so forth. These recommendations can be found in 5.2.5.

The measurements done of the effects of the implemented changes in a team in Indpro shows that the suggestions in this thesis had a positive impact on the results of the team. As stated previously, a more detailed study has to be performed to confirm that this is the case. However, in general, the changes have had positive effects and have received much positive feedback both from the developers and from the management in Indpro.

7.2 General conclusions

The general conclusions that can be extracted from the results here are those that have been handled by many other articles before, such as Höfner and Mani [8]. These articles have been mentioned a number of times throughout this thesis, and they all handle the importance of stable processes to ensure high quality.

The results of the process in the development team in Indpro clearly showed that they benefitted from a formalized development process. This process was developed to fit their
specific environment but used ideas from other researchers. It is the author’s belief that you can not draw a general conclusion regarding the results of change or insertion of a process, that is, the process has to be tailored to fit the specific needs of the organisation if you want to have a higher chance of raising the quality.

To summarize, a general process, or quality assurance routine, can most probably not be constructed for all types of environments. Instead, one has to consider the specific needs of the organisation and tailor a process or routine to fit their needs. Preferably the construction of this routine or process is guided by well founded research.

7.3 Future Recommendations

This section contains further recommendations for Indpro. They are pushed to the future because they either do not fit in the time span of the thesis or because they are not economically feasible at the moment. What the reader should also keep in mind when reading this is that what is listed is purely suggestions and recommendations that might improve future results. They have not been part of any analysis in the thesis. Most of the recommendations here are obvious flaws and parts that are missing in the current settings, of course, there might be other (better) solutions than those described here.

7.3.1 Outsourcing Environment

Requirements

As described earlier, Indpro has some problems with the requirements specifications. To ensure requirements of high quality, a requirement engineer can be hired by Indpro. He or she should then develop the requirements specifications to be used by Indpro. It would be best if this engineer also had domain knowledge regarding the most common areas that Indpro work in. This requirement engineer should also try to develop a standard requirements specification layout to be used for all specifications by Indpro, to lessen the risk that the developers misunderstand the requirements because they are given in a new format (or similar issues).

Communication

With the suggested solutions in place in all the projects in Indpro, the more basic communication issues should be solved. However, there still remain many issues regarding communication.

To ensure better communication in the future, video conferenceing with the clients would be preferrable and recommended according to research such as that conducted by Höfner and Mani [8]. To further increase the knowledge regarding Swedes and Indians, a formal training program regarding such issues should be set up. The program has to
manage both the Indians and the Swedish customers since both has to increase their understanding for each other.

The training program could for example, be handled by one of the managers in India (to train the developers) and one manager from Sweden to train the customers. They would have to construct their own training programs.

Another solution could be searching for available courses regarding such issues and send developers/customers to these. The negative part of this is that the training would not be custom made for Indpro.

7.3.2 Development Process

After the implementation of the suggestions in this thesis, Indpro has a development process that has been proved useful for a team in their development settings. However, to make sure this development process stays up to date, it needs to be continuously updated to fit new projects and ways to work in Indpro.

Hence development of this process should not stop now, instead it should be under continuous update. This can be done by, for example, the CTO. It is also crucial that all personnel in Indpro use the same process, so you can move from one team to another and quickly understand the way they work.

Test Environment

The test environment contained within the manual that was handed out to the developers is of a very basic nature (as mentioned earlier). According to the author, Indpro will most probably benefit from using the practice known as Test Driven Development (see Beck [2]). An extended evaluation to use Test Driven Development in Indpro should be done. It gives a proven advantage in iterative development and should hence be useful here.

What is most important here is to raise the general testing skill among the developers. It is the author’s belief that Test Driven Development could help with this since it introduces a new way of thinking when writing tests.

Another issue that should be brought up for evaluation is the hiring of test personnel. According to Rompaey and Demeyer [15], this helps to improve the quality of the software since this personnel will be completely dedicated to finding bugs and errors. This would also remove a task from the customer, stating the obvious bugs (which then should be found by the test personnel). The test personnel could also help train the developers in test practices and hence lessen the CTO’s work load.
References


A.1 Development Process
Quality Assurance and Test Environment – For Indpro developers, recommendations for basic Quality Assurance through testing and coding standards.
### Document modifications:

<table>
<thead>
<tr>
<th>Date</th>
<th>Modifier</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>08-05-02</td>
<td>Ronny Roos</td>
<td>Document established</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Chapters – Main goals, general coding guidelines</td>
</tr>
<tr>
<td>08-05-05</td>
<td>Ronny Roos</td>
<td>Chapters – Communication</td>
</tr>
<tr>
<td>08-05-06</td>
<td>Ronny Roos</td>
<td>Chapter – Software Development</td>
</tr>
<tr>
<td>08-05-09</td>
<td>Ronny Roos</td>
<td>Chapter – Standard test tools for different languages</td>
</tr>
<tr>
<td>08-05-10</td>
<td>Ronny Roos</td>
<td>Chapters – Documentation, Revision handling, Review process, Release process, Summary</td>
</tr>
<tr>
<td>08-05-29</td>
<td>Tom Bergström</td>
<td>Reviewed document and added suggested changes/additions</td>
</tr>
<tr>
<td>08-05-31</td>
<td>Ronny Roos</td>
<td>Modified document according input from CTO and Process Manager’s suggestions.</td>
</tr>
<tr>
<td>08-06-01</td>
<td>Ronny Roos</td>
<td>Added Recognitions, Code Review. Modified General Coding Guidelines</td>
</tr>
<tr>
<td>08-06-02</td>
<td>Ronny Roos</td>
<td>Modified General Coding Guidelines, Release Process</td>
</tr>
<tr>
<td>08-06-16</td>
<td>Ronny Roos</td>
<td>Modified .NET name standards</td>
</tr>
</tbody>
</table>
Table of content

- Document modifications: .......................................................................................................... 2
- Main goals ................................................................................................................................ 5
- General Coding Guidelines ....................................................................................................... 6
  - Object Oriented Programming ................................................................................................. 6
  - Regarding formatting ................................................................................................................ 6
  - Formatting of Code ..................................................................................................................... 6
    - Indentation ............................................................................................................................... 8
    - Function and variable names ................................................................................................... 8
    - Code spacing ........................................................................................................................... 9
  - Code commenting ...................................................................................................................... 9
    - Class and function comments ................................................................................................ 9
    - General code commenting ..................................................................................................... 9
- Team communication .................................................................................................................. 10
  - Member to member communication ......................................................................................... 10
  - Member to project leader communication ............................................................................... 10
- Customer communication .......................................................................................................... 11
  - Customer contact ..................................................................................................................... 11
- Development process .................................................................................................................. 12
  - Open your SCRUM task (Open task in Sharepoint) ............................................................... 12
  - Think before you code (Solution design) .................................................................................. 12
  - Develop your code ................................................................................................................... 12
    - Set up your development environment ................................................................................. 12
    - Next step – coding? ............................................................................................................... 13
    - Coding, unit testing, code commenting and documentation ............................................ 13
  - Manual testing ......................................................................................................................... 22
  - (Possible) Bug fixing ............................................................................................................... 22
  - Code review ............................................................................................................................. 22
  - (Possible) Code cleanup .......................................................................................................... 22
  - Documentation ......................................................................................................................... 23
  - Upload to test server and/or commit to SVN ......................................................................... 23
  - Close task in Sharepoint ......................................................................................................... 23
  - Review before release ............................................................................................................. 23
- Standard test tools for different languages .............................................................................. 24
  - Overall description of test tools ............................................................................................ 24
1 Main goals

The main goal for Indpro is to develop software with a standard satisfactory to our customers.

To do that, we need to follow a general coding standard, which implies to all developers. The standard described in this document will not go in to specific details, regarding indentation or formatting of code from different languages.

This document will contain all the general guidelines and a description of the mindset a developer of Indpro needs to have.

This document will suggest tools for the developers to use when testing their software, as well as give suggestions of how to make development procedures more efficient.
2 General Coding Guidelines
This chapter contains brief descriptions of general guidelines for coding in Indpro. The descriptions are language independent and will not contain any specifics for any particular language.

2.1 Object Oriented Programming
Object Oriented Programming or OOP is not just a way of programming. It is a way of thinking. Compared to procedural languages and function based languages; it is much easier to construct logical models of the problems you want to solve. This in turn means that it is much easier for everyone else who reads your code to understand the purpose of it, since they can logically follow a model that you have built up. An application made properly with OOP also has a much higher maintainability and it is also easier to track bugs and find other more serious errors since the programs follow a structure rather than just a sequence or a series of intertwined function calls. The advantages of OOP are many; these are just some of them, that is why Indpro has selected to work with it. We expect all the code developed here to follow these rules to as high extent as possible.

2.2 Regarding formatting
Many developers think of formatting code as a waste of time. In fact, it is not. If everyone formats their code in the same way, the code will be easier for everyone to read since they know what it looks like. Developers will save time when studying someone else’s code, they can improve their speed of reading and understanding code if they don’t have to be annoyed about why someone didn’t use (for example) the correct indentation in a particular if-case. So, the main reason for formatting code is to help each other in the team. Help yourself, help your team members, and format your code properly.

2.3 Formatting of Code
Below you will find a piece of well commented and well structured PHP-code. This presents the general formatting and commenting that should be done. Don’t mind the PHP specific parts. What one has to keep in mind when developing the code here is that the customer often have requests for how the code is supposed to look like. If this is the case, you have to follow these rules, and disregard these general guidelines.

Java example:
/**
 * Returns the total time for a contestant on a specified stage
 * @param startnr The contestant's start number
 * @param stageNr Number of the stage
 * @return Time on this stage
 */
public Time getStageTime(int startnr, int stageNr) {
    if (database.containsKey(startnr)) {
        return database.get(startnr).getStageTime(stageNr);
    } else {
        return null;
    }
}
2.3.1 Indentation
Code shall be indented according to the standard of each language.

But a general good rule to follow here is that you are supposed to increase indentation (that is, “tab in”), to demonstrate a change in code complexity (depth). That is, for an example, after if or while statements.

This is supposed to be done by using the tab-button, not using spaces. If the IDE translates the tab-press to spaces internally, let it do that, but you, as a developer should always use the tab-button.

2.3.2 Function and variable names
The naming convention is mostly decided by the customer, but, if it isn’t, use the standard for the language that you are currently writing in. If you do not know where to find this information, ask your technical project leader, he/she will know.

For an example, you are developing an application in Java, and your client asks you to include detailed Javadoc in the code. You have no idea of what the standards for Javadoc look like, if you do not find the specifications after a short search for them on the Internet, ask your project leader, he will surely know where to find it.

There are different naming conventions and code conventions for the languages used in Indpro, the following is recommended:

- **PHP**: No defined standard, depends on project.
2.3.3 Code spacing
Try to use spaces in your code where it looks “crowded” to increase readability for other developers.
Also try to use empty lines to create more room in the code where needed.

2.4 Code commenting

2.4.1 Class and function comments
The code shall always be commented according to the given standard for the language that you are writing in, in general it is more or less the same for all languages, with smaller modifications for special cases depending on the language’s special functions or way to handle things.
In general, when commenting a class and its functions, you shall include a description of the parameters, a description of the function/class itself and last but not least the return parameter and any exceptions or other unexpected things that can be generated.

2.4.2 General code commenting
General code commenting shall be done on the places in the code where there might be a chance that another developer can’t follow your train of thought.
Meaning that you have to explain to him/her what you did, through comments in the code.
Examples of this can be:
- Comparison to “magic numbers”.
  For example:

```java
if (result == 24) break;
```

This needs some explaining, since 24 has no obvious purpose here.
- Difficult parts of the code, for an example in a complex algorithm.
- In code where you included non standard libraries, this should be mentioned according to the standards described in chapter 7.
3 Team communication

Communication is one of the core components in software development. A team that does not communicate internally or externally is a team that does not function properly.

If there is no communication, the customer will not know what is happening, the project leaders will not know what is happening and most important, the team will not know what is happening.

3.1 Member to member communication

Communication within the team is essential. If there is lack of communication within a team, the members will soon get disoriented, and lose track of where the project is heading.

Team communication can be everything from official meetings, within the teams to a chat over a cup of coffee.

Take the time you need to discuss the issues with your teammates, time spent discussing an issue is not wasted time, you can often make it up later since you then have a better view of the problem at hand.

3.2 Member to project leader communication

The project leader is to be regarded as part of the team. He or she is not there to just monitor your work. They are there to help the developers perform their best.

The communication with the project leaders is best handled direct, if you have a question, ask it. They will do their best to help you with the issue or answer your question.

The project leaders are there to help you with all the aspects of working together in a team, may it be social, technical or any other aspect you can think of, they will do their best to resolve your issue.

So don’t be afraid to speak your mind or ask questions to your project leaders.
4 Customer communication

Nothing can be more effective than knowing a customer’s thoughts and values when making estimates in a development project. If you know what the customer thinks and how he or she feels about things, it will be much easier for you to understand exactly what is to be developed, and hence you can make a better estimation of the task at hand.

A good understanding of the customer also leads to a better developer-customer relationship, which is essential for effective and productive development.

4.1 Customer contact

When you, as a developer, are going to contact the customer, there are a few things you should keep in mind:

- If the issue is urgent, initiate contact via Skype/mail/phone to get a faster response. Be sure to know exactly what you want to ask the customer at this point, and be prepared to give a good description of the situation.
- If it is only a minor issue, which does not require the customer’s immediate attention, you should use Team Discussion in Sharepoint. Post your issue, and formulate yourself accurately, give examples if that is relevant for the problem at hand.
- If the issue can wait until the next meeting, don’t bring it up until then. For the meetings with the customer, make sure that you have compiled all the questions that you want to ask him/her, and all other relevant information that you want to relay to the customer before the meeting.

“Communication leads to community, that is, to understanding, intimacy and mutual valuing.”
- Rollo May
5 Development process
It is a proven fact that a well defined software process helps companies improve both their code quality and their development speed.

Our development process is what makes the code Indpro. If the code has been produced with our development process, it is approved by us, and has a quality that is satisfying for us and our customers.

For the code to be approved, it has to go through certain steps, which are defined in a checklist to be used when coding. The checklist is a compilation of the following steps.

5.1 Open your SCRUM task (Open task in Sharepoint)
Indpro is using SCRUM to control the status of the development. To make your project leaders, customers and everyone else involved in the project aware that you have started to work on a specific task you have to open the task in Sharepoint.

To do this, log in to your Sharepoint account on http://sharepoint.indpro.se/projectname. If you have not received a Sharepoint account, contact your project leaders.

When you are logged in, simply go to the current sprint, select the task you are going to start and set it to "Work in progress”.

5.2 Think before you code (Solution design)
To develop code with the best quality possible, it is necessary to plan ahead.

Before you start coding you should always make a “roadmap” for the work ahead. The map may consist of a checklist (which Indpro has approved), UML-diagrams or other sketches of what is to be built.

This is vital when constructing high quality code, which is our goal. The main point here is not that you should make a UML-diagram for every little task you have to solve. The idea is that you should have a rough sketch prepared for your task.

5.3 Develop your code
Now the actual work is about to start, but there are a few important things a developer needs to know to make the code Indpro. First of all, there are different kinds of code. A few of them are:

- Bad code
- Uncommented code
- Untested code
- Standard code
- Indpro code

We are of course, aiming for the latter. And the definition of Indpro code is:

Well commented, well structured, well documented and properly tested code.

Now you might ask yourself “How do I develop this code?”, well, the answer is easy. You read and understand the content of this document. The rest of this chapter will be dedicated to instructing you in how to specifically write high quality code.

5.3.1 Set up your development environment
This is a very important step in the process, especially if you have several team members.

To make an efficient development environment, the team should decide upon a common IDE to use for the development.
If necessary get a common storage area for the project on the server, and get all team members proper access right for this.

You should also notify your project leader about all of these decisions so he/she can keep track of the project.

5.3.2 Next step – coding?
By now you might be thinking that the next step in the process has to be coding. Your fingers are itching and you want to get to do what you do best – coding. Well, it is time to start coding, but, we are not just going to write code. At the same time we are going to create documentation, unit tests and proper comments for the code.

The actual next step is stated below.

5.3.3 Coding, unit testing, code commenting and documentation
Yes, it’s correct; all these steps will be carried out at once.

This may seem strange at first, but it actually serves a purpose. We will take you through the process of developing a simple Java-class using this method. That way you can get a more realistic example of how it works, instead of just theory.

First off, task definition:

**Develop a class for a bank account that can handle withdrawals and deposits.**

Ok, so now we have our task, what is the next step?

Design of the class of course! We will provide you with a rough sketch of what we have in mind to develop (this is enough as a design and/or roadmap).

**Design for the BankAccount class:**

```
BankAccount
+
+BankAccount(int)
+getAccountNbr() - int
+withdraw (int) - boolean
+deposit (int) - void
+getBalance() - int
```

So, now we have a rough sketch for the work that we have to do to construct this class. The next step is to start the actual coding.

Ok, so, the first step in developing this simple class might be the following:

```
public class BankAccount {
    private int accountNbr;
    public BankAccount (int accountNbr) {
        this.accountNbr = accountNbr;
    }
}
```

Alright, now we have the first functionality of this simple class in place, now we have to prove that this functionality actually works.

How do we do that? The answer is simple – Unit Testing. In Java development Indpro’s standard tool for this is JUnit. So, set up JUnit in your IDE (usually Eclipse), if you don’t know
how to do this, ask your project leader or other technical personnel. The next chapter will contain the standard tools for different languages.

So, the second step in the development process is actually developing the first test for the BankAccount-class. The first test might look something like this:

```java
import junit.framework.TestCase;

public class TestBankAccount extends TestCase {
    private BankAccount account;

    public void setUp() {
        account = new BankAccount(123);
    }

    public void tearDown() {
        account = null;
    }

    public void testAccountNotNull() {
        assertNotNull("Account was null after creation", account);
    }
}
```

What this test does, is the simplest thing one can test. It only checks that the object that is created in the `setUp()` is existing (not `null`). But is this enough to ensure that the bank account is created as it should?

No it isn’t. We still have no idea if the account was created with the correct account number. And to check that, we have to implement new functionality in the BankAccount-class:

```java
public int getAccountNbr() {
    return this.accountNbr;
}
```

Now we can check the account number. So let’s make sure the account has been created as it should through the following test:

```java
public void testAccountNumber() {
    // Make sure that the initial account has number 123
    assertEquals("Account number for " + account + " not correct", 123, account.getAccountNbr());

    // Create a new account and make sure that it gets the right number
    BankAccount ba1 = new BankAccount(0);
    assertEquals("Account number for " + ba1 + " not correct", 0, ba1.getAccountNbr());

    // Create a new account and make sure that you can create an account with a large number
    BankAccount ba2 = new BankAccount(1234567890);
    assertEquals("Account number for " + ba2 + " not correct", 1234567890, ba2.getAccountNbr());
}
```

The next step here is to take a step back and look at the code developed and ask yourself a few questions:

- Is my code well commented?
- Is the code properly formatted?
- Can someone else understand my code?
- Is the code properly documented?
- Does the code need refactoring?
In this case, the answer would be:

- In some places it is, but comments are missing.
- Yes it is.
- Very likely.
- No it is not.
- No it does not.

Meaning that we have a few things to fix. So, after fixing these things, the code looks like this:

```java
public class BankAccount {
    private int accountNbr;
    /**
     * Creates a new bank account with a specified account number
     * @param accountNbr The specified account number
     */
    public BankAccount(int accountNbr) {
        this.accountNbr = accountNbr;
    }
    /**
     * Retrieves the account number.
     * @return The account number
     */
    public int getAccountNbr() {
        return this.accountNbr;
    }
}
```
Test class:

```java
import junit.framework.TestCase;

class TestBankAccount extends TestCase {

  // The bank account on which many of the tests will be done
  private BankAccount account;

  /**
   * setUp function for JUnit
   */
  public void setUp() {
    account = new BankAccount(123);
  }

  /**
   * tearDown function for JUnit
   */
  public void tearDown() {
    account = null;
  }

  /**
   * Tests that accounts are actually created.
   */
  public void testAccountNotNull() {
    assertNotNull("Account was null after creation", account);
  }

  /**
   * Tests different aspects of the account number
   */
  public void testAccountNumber() {
    // Make sure that the initial account has number 123
    assertEquals("Account number for " + account.getName() + " not correct", 123, account.getAccountNbr());

    // Create a new account and make sure that it gets the right number
    BankAccount ba1 = new BankAccount(0);
    assertEquals("Account number for " + ba1.getName() + " not correct", 0, ba1.getAccountNbr());

    // Create a new account and make sure that you can create an account
    // with a large number
    BankAccount ba2 = new BankAccount(1234567890);
    assertEquals("Account number for " + ba2.getName() + " not correct", 1234567890, ba2.getAccountNbr());
  }
}
```

Ok, so, now we have fixed all the questions, except the one regarding technical documentation. This code is enough to generate JavaDoc, which is a form of technical documentation. But if your customer has requested specific documentation for his/her project, now is the time to create that.

Before moving on, you should ask yourself one final question:

- Is my code properly tested (e.g. is my developed code good enough to release to customer?)?

In this case the answer would be no, because there is one obvious flaw in the tests. What happens if you try to initiate an account with a negative number? This is nothing that is neither checked, nor blocked. And what happens if you call the class with another input than an integer. These are all things to be considered if this was a class to be used in a program.
where users enter data, since you then most probably would have to handle exceptions and block certain types of input.

What parts of this should then be handled in the unit tests?

Well, that is an area that has been discussed a lot, and we have chosen to stay with the parts that actually involve the purpose of the code that is being developed. I.e. checking for invalid input in the bank account class (such as inserting a string instead of an integer) does not belong here but should be handled elsewhere in the code, for an example in the layer of code just beneath the GUI, where you handle user input.

What should be tested here on the other hand is that you can have an account with a negative number. Even though you can’t have this in the real world, we can in our class. If you want to block this, it is recommended that you do this among the other input validation functions (which should all be tested).

So, now that we have found out that we didn’t test our functionality enough, we add a new assertEquals():

```java
// Test negative account number
BankAccount ba3 = new BankAccount(-123);
assertEquals("Account number for \"ba3\" not correct", -123, ba3.getAccountNbr());
```

So, what have we achieved here? We have produced less code than we usually do, with more delays and problems than are strictly necessary.

But what we have also done is produced code that is ready for release to customer or management within seconds, all you have to do is build the application, build the documentation, package it and send it to whoever requested it (and this can all be done with one mouse click with for an example Ant scripts).

We no longer have to spend weeks on commenting code that we developed six months ago, and we no longer have to put in that extra month in our projects to do the technical documentation requested by the customer, because we do that all the time, while we code. Not only does this save time and money, it also saves effort, since we are more motivated during development than when we are forced to sit weeks doing nothing but technical documentation.

The next step is to, once again, develop new functionality. The next step that we have chosen is to implement the account balance and the `withdraw()` and `deposit()` functions. That is, the rest of the class.
**Step 1: Develop functionality**

*BankAccount class [new functionality]:*

```java
public class BankAccount {

  private int balance;

  /**
   * Creates a new bank account with a specified account number
   * Initially the account balance is set to 0
   * @param accountNbr The specified account number
   */
  public BankAccount(int accountNbr) {
    this.accountNbr = accountNbr;
    this.balance = 0;
  }

  public boolean withdraw(int amount) {
    if (balance >= amount && amount > 0){
      balance -= amount;
    }
    return balance >= amount && amount > 0;
  }

  public void deposit(int amount) {
    if(amount > 0) {
      balance += amount;
    }
  }

  public int getBalance() {
    return balance;
  }
}
```
**Step 2: Test functionality**

**Test class [new tests]:**

```java
public class TestBankAccount extends TestCase {

    public void testDeposit() {
        // Deposit 1000 and check that the balance is 1000
        account.deposit(1000);
        assertEquals("Account balance incorrect", 1000, account.getBalance());

        // Deposit negative number, check balance = 1000
        account.deposit(-900);
        assertEquals("Account balance incorrect", 1000, account.getBalance());

        // Deposit 100, check balance = 1100
        account.deposit(100);
        assertEquals("Account balance incorrect", 1100, account.getBalance());

        // Another deposit
        account.deposit(4);
        assertEquals("Account balance incorrect", 1104, account.getBalance());
    }

    public void testWithdraw() {
        // Set up an account with 900 so we can run some tests
        account.deposit(900);

        // Make sure we can't withdraw more than is available
        assertFalse("Withdrawal possible even though it shouldn't be",
                    account.withdraw(1000));

        // Make sure we can withdraw less than what is available
        assertTrue("Withdrawal not possible even though it should be",
                    account.withdraw(100));

        // Make sure the correct amount was withdrawn
        assertEquals("Incorrect funds withdrawn", 800, account.getBalance());

        // Make sure you can't withdraw negative amounts
        account.withdraw(-100);
        assertEquals("Negative withdrawal succeeded", 800, account.getBalance());
    }
}
```

**Step 3: Step back, refactor, comment and document the code**

**BankAccount class finished:**

```java
public class BankAccount {

    private int accountNbr;
    private int balance;

    /**
     * Creates a new bank account with a specified account number
     * Initially the account balance is set to 0
     * @param accountNbr The specified account number
     */
    public BankAccount(int accountNbr) {
        this.accountNbr = accountNbr;
        this.balance = 0;
    }
}
```
/**
 * Retrieves the account number.
 * @return The account number
 */
public int getAccountNbr() {
    return this.accountNbr;
}

/**
 * Withdraws money from the account
 * @param amount The amount to withdraw
 * @return false if there is insufficient funds or invalid amount
 */
public boolean withdraw(int amount) {
    if (balance >= amount && amount > 0){
        balance -= amount;
    }
    return balance >= amount && amount > 0;
}

/**
 * Deposits money to the account
 * @param amount Amount to deposit
 */
public void deposit(int amount) {
    if(amount > 0) {
        balance += amount;
    }
}

/**
 * Gets the account balance
 * @return Balance
 */
public int getBalance() {
    return balance;
}

Test class finished:
import junit.framework.TestCase;
public class TestBankAccount extends TestCase {
    // The bank account on which many of the tests will be done
    private BankAccount account;
    /**
     * setUp function for JUnit
     */
    public void setUp() {
        account = new BankAccount(123);
    }
    /**
     * tearDown function for JUnit
     */
    public void tearDown() {
        account = null;
    }
}
/**
 * Tests that accounts are actually created.
 */
public void testAccountNotNull() {
    assertNotNull("Account was null after creation", account);
}

/**
 * Tests that deposit works as it should.
 */
public void testDeposit() {
    // Deposit 1000 and check that the balance is 1000
    account.deposit(1000);
    assertEquals("Account balance incorrect", 1000, account.getBalance());

    // Deposit negative number, check balance = 1000
    account.deposit(-900);
    assertEquals("Account balance incorrect", 1000, account.getBalance());

    // Deposit 100, check balance = 1100
    account.deposit(100);
    assertEquals("Incorrect funds withdrawn", 1100, account.getBalance());

    // Another deposit
    account.deposit(4);
    assertEquals("Account balance incorrect", 1104, account.getBalance());
}

/**
 * Tests the withdraw function
 */
public void testWithdraw() {
    // Set up an account with 900 so we can run some tests
    account.deposit(900);

    // Make sure we can't withdraw more than is available
    assertFalse("Withdrawal possible even though it shouldn't be", account.withdraw(1000));

    // Make sure we can withdraw less than what is available
    assertTrue("Withdrawal not possible even though it should be", account.withdraw(100));

    // Make sure the correct amount was withdrawn
    assertEquals("Incorrect funds withdrawn", 800, account.getBalance());

    // Make sure you can't withdraw negative amounts
    account.withdraw(-100);
    assertEquals("Negative withdrawal succeeded", 800, account.getBalance());
}

/**
 * Tests different aspects of the account number
 */
public void testAccountNumber() {
    // Make sure that the initial account has number 123
    assertEquals("Account number for \"account\" not correct", 123, account.getAccountNbr());

    // Create a new account and make sure that it gets the right number
    BankAccount bal = new BankAccount(0);
    assertEquals("Account number for \"bal\" not correct", 0, bal.getAccountNbr());

    // Create a new account and make sure that you can create an account
    // with a large number
    BankAccount ba2 = new BankAccount(1234567890);
So, now we have gone through this entire process and finished our class, now we have to be done?

Well, no, we are not. We have some of the most important steps left.

5.4 Manual testing
We have to put our product in the right context (use it in a GUI, put it on a web server and run it, etc.), use it, and do our best to break it.

This can be done by for example (standard examples)

- Inputting text in boxes only intended for numbers
- Inputting negative values in boxes intended for positive
- Inputting other “border values” (very large numbers, very small numbers)
- Inputting variations of text and numbers
- Testing items in your program that you know are going to fail, and make sure the errors are handled correct (exceptions and so forth)

The above are just simple standard examples of what you can do to test your program. You have to think for yourself and try to figure out all the different things that a user might input in the system. Your goal here is not to act as the developer you actually are, but instead, to act as the worst system user ever, which might input things you couldn’t even imagine in your system.

5.5 (Possible) Bug fixing
By now, you have probably broken your system in various ways. If you haven’t found at least one bug in the manual testing, it is strongly recommended that you go back to the manual testing and rethink your input to the system.

It is now time to fix these bugs. What you should consider here is that, you in the first place did not do this correctly, meaning that it is probably not an obvious thing in the code. So, here it is strongly recommended that you put extra care into commenting the code that you develop to fix the bugs found.

5.6 Code review
This is one of the tasks that are very important in the beginning. Once again, after all the bug fixing and system crashing, it is time to take a step back and review your own code, critically.

There are a few questions you should ask yourself when doing this:

- Is my code properly commented?
- Is my code properly documented?
- Is my code properly formatted?
- Can another programmer understand my code?
- Is there enough technical documentation for another programmer to understand the entire system?
If the answer to all these questions is yes, your code passes the code review. If you find yourself in doubt about one or more of the questions, fix the things that you are not sure about. If you are very insecure about your decision, ask your project leader to review the code with you, or ask him/her to help you answer some of the specific questions above. Further information about conducting code reviews can be found in chapter 9.

5.7 (Possible) Code cleanup

After the review it is, of course, time to do something about the things that you found unsatisfactory in your code.

When performing this step, don’t forget to properly comment and document all changes you make.

5.8 Documentation

If the customer has requested some kind of documentation or user manual for the project, now is the time to create this for the particular functionality that was just developed.

Be sure to express yourself in a proper way, and attach screenshots or similar where such information might be needed.

5.9 Upload to test server and/or commit to SVN

This is one of the more important items in the development process. To get continuous progress in the developed code for the system, everyone in the team should make a “commit” to the repository as often as possible.

As a rule, once you have completed new functionality and gone through all these steps, make a commit, and everyone else can get the new code when they update.

This way, no one has to handle major merges in the project.

*It is strongly recommended to read the following regarding revision handling in software development:*


5.10 Close task in Sharepoint

We are to perform more or less the same thing as described in the above subchapter “Open task in Sharepoint”, but now we are going to set the status to “Finished” (or similar) instead.

5.11 Review before release

Before the release a review is conducted. This review is to be done by both you and the project supervisor.

The review shall be performed according to chapter “Review process” below, what you should also keep in mind is that a review can also be conducted with the customer, to show him/her what has happened in the project. When performing a review with a customer, a so called demonstration, or other form of review, be sure to be well prepared.
6 Standard test tools for different languages
This section will contain a description of the test tools to use for different languages when developing for Indpro. If the customer has not specified anything else, these are the tools you, as a developer, are to use.

6.1 Overall description of test tools
The test tools that we use in the development process are simply there to help us ensure ourselves that what we have produced is of good enough quality.

This can be assured by writing unit tests (of high quality that tests the correct parts of the code) and also, to some extent, by making sure that the test coverage is high enough on the written code.

6.2 .NET
In .NET we will use the following tools:

  - The tests are still written in the exact same way as they are in NUnit.

6.2.1 Usage of TestMatrix
First of all, you need to download and install the component from the link above.

When you have done this, start the edition of Visual Studio you use, and you should see some additions to the standard tools.

To the left is a screenshot of the TestMatrix menu. This menu is inserted among all the standard menus of Visual Studio.

There are options to run all tests in the solution, the project and just single tests. These options also exist when right clicking a project/test class or solution.

It is recommended to learn some shortcuts since you will be running the tests frequently.
This is a dockable window that displays the result of the test runs. As you can see, there are two different windows for the results of the test and the test coverage.

It is recommended that you place these windows where you see fit, since you will be working a lot with them.

Above is the result of the coverage of the different projects.

Below follows the same example in .NET as was created above in Java.

**BankAccount class file:**

```csharp
using System;

namespace BankAccount
{
    /// <summary>
    /// A class to handle basic operations on a bank account
    /// </summary>
    public class BankAccount
    {
        private int balance;
        private int accountNbr;

        /// <summary>
        /// Constructor for the BankAccount, takes the account number as parameter
        /// </summary>
        /// <param name="accountNumber">The account number</param>
        public BankAccount(int accountNbr)
        {
            this.accountNbr = accountNbr;
            this.balance = 0;
        }

        /// <summary>
        /// Returns the account number for the current account
        /// </summary>
    }
}
public int GetAccountNbr()
{
    return this.accountNbr;
}

/// <summary>
/// Withdraws a specified amount of money from the account
/// </summary>
/// <param name="amount">The amount of money to withdraw</param>
/// <returns>False if withdrawal not possible, true otherwise</returns>
public bool Withdraw(int amount)
{
    if (balance >= amount && amount > 0)
    {
        balance -= amount;
    }
    return balance >= amount && amount > 0;
}

/// <summary>
/// Deposits a specified amount of money
/// </summary>
/// <param name="amount">The amount to deposit</param>
public void Deposit(int amount)
{
    if (amount > 0)
    {
        balance += amount;
    }
}

/// <summary>
/// Get the account balance
/// </summary>
/// <returns>The balance</returns>
public int GetBalance()
{
    return balance;
}
}

BankAccount test class:

namespace BankAccount
{
    using System;
    using NUnit.Framework;

    /// <summary>
    /// Tests to assure the functionality of the bank account.
    /// </summary>
    [TestFixture]
    public class TestBankAccount
    {
        /// <summary>
        /// The bank account on which most tests will run
        /// </summary>
        private BankAccount account;

        /// <summary>
        /// The SetUp function for NUnit
        /// </summary>
        [SetUp]
        public void Init()
        {
            account = new BankAccount(123);
        }

        /// <summary>
        /// The TearDown function for NUnit
        /// </summary>
    }
}
public void Dispose()
{
    account = null;
}

/// <summary>
/// Test account creation
/// </summary>
///
[Test]
public void TestAccountNotNull()
{
    Assert.IsNotNull(account, "Account is null after creation");
}

/// <summary>
/// Tests that the account got the correct account number
/// </summary>
///
[Test]
public void TestAccountNumber()
{
    // Check that the original account holds the correct account number
    Assert.AreEqual(123, account.GetAccountNbr(), "Account numbers not matching - account");

    // Create new accounts and insert borderline/and or other important values
    BankAccount ba1 = new BankAccount(1234567890); // large integer
    BankAccount ba2 = new BankAccount(0); // 0-value
    BankAccount ba3 = new BankAccount(-123); // negative value
    Assert.AreEqual(1234567890, ba1.GetAccountNbr(), "Account numbers not matching - ba1");
    Assert.AreEqual(0, ba2.GetAccountNbr(), "Account numbers not matching - ba2");
    Assert.AreEqual(-123, ba3.GetAccountNbr(), "Account numbers not matching - ba3");
}

[Test]
public void TestDeposit()
{
    // Deposit 1000 and check that the balance is 1000
    account.Deposit(1000);
    Assert.AreEqual(1000, account.GetBalance(), "Account balance incorrect");

    // Deposit negative number, check balance = 1000
    account.Deposit(-900);
    Assert.AreEqual(1000, account.GetBalance(), "Negative deposit possible");

    // Deposit 100, check balance = 1100
    account.Deposit(100);
    Assert.AreEqual(1100, account.GetBalance(), "Failed to deposit correct amount");

    // Another deposit
    account.Deposit(4);
    Assert.AreEqual(1104, account.GetBalance(), "Failed third deposit");
}

[Test]
public void TestWithdraw()
{
    // Set up an account with 900 so we can run some tests
    account.Deposit(900);

    // Make sure we can't withdraw more than is available
    Assert.IsFalse(account.Withdraw(1000), "Withdrawal possible even though it shouldn't be");

    // Make sure we can withdraw less than what is available
To run the tests, simply press the TestMatrix menu, and push “Run project tests” (or “Run solution tests” depending on which one you would like to do).

When that is done, you will get the results in the two windows, one with coverage, and the other one with test results.

To get the tests to work, you need working references to NUnit in your project (you can also use the bundled NUnit that comes with TestMatrix).

### 6.3 Java

In Java we will use the following tools:

- **JUnit**
  - Comes integrated with Eclipse, and can be found here: [http://www.junit.org/](http://www.junit.org/)
- (If using Eclipse) Coverlipse

As you know by now, coverage is just a measurement for how much of the code is executed during the tests. Since Coverlipse is not something that everyone that is working in java can use, we are focusing on the description of JUnit. An example of testing in JUnit can be found above.

Since we are not using a mandatory coverage tool for Java, it is very important that you are sure that you have tested all the possible functionality.

#### 6.3.1 Testing with JUnit

So, when you have written your tests according to the above model, you of course want to run them and get the result.
This is how you do it in Eclipse:
Right click your project in the Navigator, select Run As -> JUnit Test

After the tests have run, you will get the result, if everything is as it should be, you will see this window:
If there are errors in your test you will get something like the following, and a failure trace at the bottom:

If you are running JUnit as a standalone, it works more or less the same. You load the test classes you want to run, and the result is displayed in the same manner.

6.4 PHP

When it comes to PHP, unit testing is relatively new. We have chosen to work with a framework called SimpleTest.

SimpleTest can be downloaded from here: http://simpletest.sourceforge.net/

That site also contains documentation for how to use it. Since how to run and how to set up this environment will vary very much depending on what project you are going to use it in. It will not be discussed here.

Instead, you can find an example class and an example test below. Don’t feel restricted to this example, but feel free to use the framework in other ways. What is important is that the functionality of the developed classes is tested.

What you also especially need to keep in mind when using PHP-unit testing is that you pay special attention to always run all previous tests, so that you make sure you did not break anything with the new code.

This is documented here:
BankAccount class file:

```php
<?php
/**
 * A simple class to handle a bank account
 */
class BankAccount {
    private $accountNbr;
    private $balance;
    /**
     * Constructor for the account
     * @param int $accountNbr
     */
    public function __construct($accountNbr) {
        $this->accountNbr = $accountNbr;
        $this->balance = 0;
    }
    /**
     * Returns the account number
     * @return The account number
     */
    public function getAccountNbr() {
        return $this->accountNbr;
    }
    /**
     * Withdraws money from the account
     * @param amount The amount to withdraw
     * @return false if there is insufficient funds or invalid amount
     */
    public function withdraw($amount) {
        if ($this->balance >= $amount && $amount > 0) {
            $this->balance -= $amount;
        }
        return $this->balance >= $amount && $amount > 0;
    }
    /**
     * Deposits money to the account
     * @param amount Amount to deposit
     */
    public function deposit($amount) {
        if ($amount > 0) {
            $this->balance += $amount;
        }
    }
    /**
     * Gets the account balance
     * @return Balance
     */
    public function getBalance() {
        return $this->balance;
    }
}
?>
```
<?php

require_once ('simpletest/autorun.php');
require_once ('BankAccount.php');

class TestBankAccount extends UnitTestCase {
    private $account;

    /**
     * Optional constructor for UnitTestCase
     */
    public function FileTestCase(){
        $this->UnitTestCase('Test of BankAccount class');
    }
    /**
     * SimpleTest setUp
     */
    public function setUp(){
        $this->account = new BankAccount(123);
    }
    /**
     * SimpleTest tearDown
     */
    public function tearDown(){
        $this->account = null;
    }
    /**
     * Tests that accounts are actually created->
     */
    public function testAccountNotNull(){
        $this->assertNotNull($this->account, "Account was null after creation");
    }
    /**
     * Tests that deposit works as it should->
     */
    public function testDeposit(){
        // Deposit 1000 and check that the balance is 1000
        $this->account->deposit(1000);
        $this->assertEqual(1000, $this->account->getBalance(), "Account balance incorrect");

        // Deposit negative number, check balance = 1000
        $this->account->deposit(-900);
        $this->assertEqual(1000, $this->account->getBalance(), "Account balance incorrect");

        // Deposit 100, check balance = 1100
        $this->account->deposit(100);
        $this->assertEqual(1100, $this->account->getBalance(), "Account balance incorrect");

        // Another deposit
        $this->account->deposit(4);
        $this->assertEqual(1104, $this->account->getBalance(), "Account balance incorrect");
    }
    /**
     * Tests the withdraw function
     */
    public function testWithdraw(){
        // Set up an account with 900 so we can run some tests
        $this->account->deposit(900);

        // Make sure we can’t withdraw more than is available
        $this->assertFalse($this->account->withdraw(1000), "Withdrawal
```
possible even though it shouldn’t be”); 
  // Make sure we can withdraw less than what is available 
  $this->assertTrue($this->account->withdraw(100), "Withdrawal not 
possible even though it should be"); 

  // Make sure the correct amount was withdrawn 
  $this->assertEqual(800, $this->account->getBalance(), "Incorrect funds 
withdrawn"); 

  // Make sure you can’t withdraw negative amounts 
  $this->account->withdraw(-100); 
  $this->assertEqual(800, $this->account->getBalance(), "Negative 
withdrawal succeeded"); 
} 

/** 
 * Tests different aspects of the account number 
 */
 
public function testAccountNumber() { 
  // Make sure that the initial account has number 123 
  $this->assertEqual(123, $this->account->getAccountNbr(), "Account 
number for "account" not correct"); 

  // Create a new account and make sure that it gets the right number 
  $ba1 = new BankAccount(0); 
  $this->assertEqual(0, $ba1->getAccountNbr(), "Account number for 
"ba1" not correct"); 

  // Create a new account and make sure that you can create an account 
  // with a large number 
  $ba2 = new BankAccount(1234567890); 
  $this->assertEqual(1234567890, $ba2->getAccountNbr(), "Account number for 
"ba2" not correct"); 

  // Test negative account number 
  $ba3 = new BankAccount(-123); 
  $this->assertEqual(-123, $ba3->getAccountNbr(), "Account number for 
"ba3" not correct"); 
} 
?>
7 Standard documentation for different languages

This chapter will contain brief information about standard documentation techniques (in the code) for different languages. The reason that it is important to follow these is that you can easily generate technical documentation in the style of your language if you have done it correctly.

For an example using JavaDoc or Sandcastle.

7.1 Java

When using Java, document your code in the standard “Java fashion” so you can generate JavaDoc from it.

A description of how to do this is found here:

http://java.sun.com/j2se/javadoc/writingdoccomments/

7.2 .NET

As standard here, we will be following the Sandcastle documentation. Sandcastle is a tool that is used to generate documentation in the MSDN fashion.

You can find a description of how to write code comments for Sandcastle here: (HOWTO for C#) http://msdn.microsoft.com/en-us/library/z04awywx(VS.80).aspx

More information can be found using the links on that site.

To generate the documentation with sandcastle the command line is used. For those that prefer a graphical interface the Sandcastle Help File Builder is available (but not produced by Microsoft). You can find that tool here: http://www.codeplex.com/SHFB

7.3 PHP

For PHP we will use the PHPODocumentor format. See the link below for more information on how to generate documentation with this tool, and how to comment your code properly.

8 Revision handling

This is a very important concept when it comes to software development. Many hours, months and even years have been spent trying to find effective ways to handle this.

To effectively handle versions of documents, modules and solutions, Indpro uses SVN, with the SVN client Tortoise SVN (http://tortoisesvn.tigris.org/).

If you want to use plugins for your development environment (for an example Subclipse for Eclipse), feel free to install them and use them, but the same usage rules apply. But, in this case, it is not the tool that is important, but the understanding about how the system works. This will not be discussed in this document; instead it is recommended that you read (at least) the two first chapters in the E-book found at http://svnbook.red-bean.com/. It is highly recommended that you read these chapters before proceeding, since the rules and recommendations are using terms that are explained in that book.

8.1 Rules and recommendations regarding SVN

- When passing a milestone and/or release, you shall tag the code in the repository with a note stating which milestone and/or release it is.
- You are going to make regular commits to the repository (meaning that you make a commit as soon as you have gone through the development process and added new functionality).
- Before you commit, you always update to the latest version from the repository.
  - Update
    - If conflicts exists, fix conflicts
  - Update
    - If conflicts exists, fix conflicts
  - And so forth, until there are no conflicts, then you finally make your commit.
- All conflicts have to be solved before committing to the repository.
- The repository must never contain unfinished code, and what is more important, must never contain code that does note compile. Keep the repository clean!
- Use the following standard layout of the repository
  - Tags (Where all the tagged versions are)
  - Branches (Where the branches are worked with)
  - Trunk (Main work area)
9 Review process

When conducting a code review, it is important that you forget all the ties that you have to the code and view it critically.

Your goal when reviewing the code is finding possible flaws, that might later pose problems (design flaws, programming faults etc.).

As a general rule, it is always good to first try to get an overview of the complete code that you are reviewing, so you can understand how the classes depend on each other, what modules have interdependencies and so forth.

From there it is a good idea to run a static test tool, to find the more basic design flaws and pure programming faults fast. These faults can then be viewed in the code and you can quickly determine problem areas based on these (or the faults found might not be problem areas, this is something that you as a reviewer have to decide).

- Static testing .NET: FxCop
- Static testing Java: FindBugs

After this quick entrance in the code, you now have to start the real review. In general, it is a good idea to start from the point of execution, that is, the methods that are called by for an example a GUI. From there you simply track the code process and review the code.

This might sound easy, but it is not. Reviewing code is difficult, especially if it is your own. Which is why, it is a general recommendation not to review your own code, since you are likely to find less errors than an external reviewer.

To make sure that you are prepared to review code, follow the steps in the chapter below.

9.1 Code review process

The following is basic points that you have to make sure the code (and you) handle:

- Do you understand the code you are reviewing?
  - Understand what the new code, code it depends on and code that depends on it does.
  - Always read and understand what each line of code does; follow the code like a debugger would have done.
  - Follow the codes through all paths, and think about how it reacts.
  - Learn what that members/classes do in a big picture, how they are talking to each other.
  - If necessary to understand the new code, also read and understand already reviewed code that exist around the new code.

- See if it’s possible to write the code better. In this case discuss your suggestion with your team members and see what conclusion you reach.
- Check if function size and complexity is unreasonable.
- Check for unclear expression of ideas in the code.
- Check for poor encapsulation.
- Check for function prototypes that are not correctly used.
- Check for data types that do not match.
- Check for uninitialized variables at start of function.
- Check for uninitialized variables going into loops.
- Check for poor logic - won’t function as needed.
- Check for error condition not caught (e.g., return codes from malloc()).
- Check for switch statement without a default case (if only a subset of the possible conditions used).
- Check for incorrect syntax - such as proper use of ==, =, &&, &,
- Check for non reentrant code in dangerous places.
  http://en.wikipedia.org/wiki/Reentrant
- Check for slow code in areas where speed is important.
- Does the code build correctly?
- Does the code execute as expected?

9.1.1 Comments and Coding Conventions
- Check that the comments match what the code is doing?
- Check that all coding conventions are followed.
- Check for poor commenting.
- Are variable declarations properly commented? 
  Comments are required for aspects of variables that the name doesn’t describe.
- Are units of numeric data clearly stated?
  Comment the units of numeric data. For example, if a number represents length, indicate if it is in feet or meters.
- Are all functions, methods and classes documented?
  Describe each routine, method, and class in one or two sentences at the top of its definition. If you find that the description of the method is too complicated to write, it might be that the function itself is too complicated and needs refactoring.
- Are function parameters used for input or output clearly identified as such?
- Are complex algorithms and code optimizations adequately commented?
  Complex areas, algorithms, and code optimizations should be sufficiently commented, so other developers can understand the code and walk through it.
- Does code that has been commented out have an explanation?
  There should be an explanation for any code that is commented out. "Dead Code" should be removed. If it is a temporary hack, it should be identified as such.
- Are comments used to identify missing functionality or unresolved issues in the code?
  TODO-comments and other common identifiers.

9.1.2 Error Handling
- Are assertions used everywhere data is expected to have a valid value or range?
  Assertions make it easier to identify potential problems. For example, test if pointers or references are valid.
- Are errors properly handled each time a function returns?
  An error should be detected and handled if it affects the execution of the rest of a routine. For example, if a resource allocation fails, this affects the rest of the routine if it uses that resource. This should be detected and proper action taken. In some cases, the "proper action" may simply be to log the error.
- Are resources and memory released in all error paths?
  Make sure all resources and memory allocated are released in the error paths.
- Are all thrown exceptions handled properly?
  If the source code uses a routine that throws an exception, there should be a function in the call stack that catches it and handles it properly.
• Is the function caller notified when an error is detected?
   Consider notifying your caller when an error is detected. If the error might affect your caller, the caller should be notified. For example, the “Open” methods of a file class should return error conditions. Even if the class stays in a valid state and other calls to the class will be handled properly, the caller might be interested in doing some error handling of its own.

• Has error handling code been tested?
   Don’t forget that error handling code can be defective. It is important to write test cases that exercise it.

9.1.3 Thread Safeness
• Are all global variables thread-safe?
   If global variables can be accessed by more than one thread, code altering the global variable should be enclosed using a synchronization mechanism such as a mutex. Code accessing the variable should be enclosed with the same mechanism.

• Are objects accessed by multiple threads thread-safe?
   If some objects can be accessed by more than one thread, make sure member variables are protected by synchronization mechanisms.

• Are locks released in the same order they are obtained?
   It is important to release the locks in the same order they were acquired to avoid deadlock situations. Check error code paths.

• Is there any possible deadlock or lock contention?
   Make sure there’s no possibility for acquiring a set of locks (mutex, semaphores, etc.) in different orders. For example, if Thread A acquires Lock #1 and then Lock #2, then Thread B shouldn't acquire Lock #2 and then Lock #1.

9.1.4 Control Structures
• Are loop ending conditions accurate?
   Check all loops to make sure they iterate the right number of times. Check the condition that ends the loop; ensure it will end out doing the expected number of iterations.

• Is the code free of unintended infinite loops?
   Check for code paths that can cause infinite loops. Make sure end loop conditions will be met unless otherwise documented.

9.1.5 Performance
• Do recursive functions run within a reasonable amount of stack space?
   Recursive functions should run with a reasonable amount of stack space. Generally, it is better to code iterative functions.

• Are whole objects duplicated when only references are needed?
   This happens when objects are passed by value when only references are required. This also applies to algorithms that copy a lot of memory. Consider using algorithm that minimizes the number of object duplications, reducing the data that needs to be transferred in memory.

• Does the code have an impact on size, speed, or memory use?
   Can it be optimized? For instance, if you use data structures with a large number of occurrences, you might want to reduce the size of the structure.

• Are you using blocking system calls when performance is involved?
   Consider using a different thread for code making a function call that blocks.

• Is the code doing busy waits instead of using synchronization mechanisms or timer events?
Doing busy waits takes up CPU time. It is a better practice to use synchronization mechanisms.

- Was this optimization really needed?
  Optimizations often make code harder to read and more likely to contain bugs. Such optimizations should be avoided unless a need has been identified. Has the code been profiled?

9.1.6 Functions
- Are function parameters explicitly verified in the code?
  This check is encouraged for functions where you don’t control the whole range of values that are sent to the function. This isn’t the case for helper functions, for instance. Each function should check its parameter for minimum and maximum possible values. Each pointer or reference should be checked to see if it is null. An error or an exception should occur if a parameter is invalid.
- Are arrays explicitly checked for out-of-bound indexes?
  Make sure an error message is displayed if an index is out-of-bound.
- Are functions returning references to objects declared on the stack?
  Don’t return references to objects declared on the stack, return references to objects created on the heap.
- Are variables initialized before they are used?
  Make sure there are no code paths where variables are used prior to being initialized. If an object is used by more than one thread, make sure the object is not in use by another thread when you destroy it. If an object is created by doing a function call, make sure the object was created before using it.
- Does the code re-write functionality that could be achieved by using an existing API?
  Don’t reinvent the wheel. New code should use existing functionality as much as possible. Don’t rewrite source code that already exists in the project. Code that is replicated in more than one function should be put in a helper function for easier maintenance.

9.1.7 Bug Fixes
- Does a fix made to a function change the behavior of caller functions?
  Sometimes code expects a function to behave incorrectly. Fixing the function can, in some cases, break the caller. If this happens, either fix the code that depends on the function, or add a comment explaining why the code can’t be changed.
- Does the bug fix correct all the occurrences of the bug?
  If the code you’re reviewing is fixing a bug, make sure it fixes all the occurrences of the bug.

9.1.8 Math
- Is the code doing signed/unsigned conversions?
  Check all signed to unsigned conversions: Can sign completion cause problems? Check all unsigned to signed conversions: Can overflow occur? Test with Minimum and Maximum possible values.
9.2 Summary

What you should also keep in mind when reviewing code is that you have to know what type of application it is you are reviewing. Is security an issue? Speed? Or perhaps memory usage? All aspects of such an application have to be considered, with extra focus on the critical parts of it.

When reviewing critical areas of code it is important that you know what to look for and that you understand how to create efficient code for that type of application.

For further review help on this type of application, please contact your project manager.

What is also important to know when using the above list is of course that the entire list is not valid for all projects. You have to know what parts to use, either you ask your project manager about it or you ask another team member what was decided internally in the team.
10 Release process
Now we have come to the final stage of the development process. Now, your code is developed, tested, and documented, and it has come to the time where you want to release it to the customer.

So, how do we do this? The answer is simple, follow the list below, and make sure that all the steps are fulfilled.

- Make sure your code is properly documented, commented and tested.
- Review your own code according to “Review process” above.
- Notify project supervisor at least 4 hours prior to the release time that it is time for release, and that your code is ready for review by him/her.
- The release is tagged in the repository according to the standards.
- The supervisor checks out the tagged version.
  - The supervisor reviews the code according to “Review process” above.
    - If there is anything urgent that needs to be fixed, it is done if there is time.
    - Otherwise, the customer is notified about the flaws found in the code during a review meeting. How the flaws are to be handled is also decided in this meeting. Here you, as a developer need to be prepared to use different tools to satisfy the customer’s need for information. Different tools to use can be:
      - Video conferencing
      - Desktop sharing
      - Prerecorded movies
      - Etc. (customers preferences)

10.1 Regarding releases
The releases in a project are what set the standard and quality of the entire project. Meaning that, the work in a team will be evaluated based on the releases they make and the quality of those releases.

One very important thing regarding releases is the deadlines for them. If you know that you are going to miss the deadline, notify the customer about this, and give a new time estimate for the release.

If you know that you will not be able to deliver what was specified in the release, cut back on the amount of items released and make a proper release instead of one with lots of bugs.

The most important thing to remember here is to always notify the customer of changes in release plans. He or she might be very dependent of the release time.

One thing you should also keep in mind here is that Indpro is mainly working with Swedish customers and some of them may be very sensitive if they don’t get any notification about their releases.

Place yourself in their shoes. You have paid for a release, containing specific content on a certain time, if that does not show up on the specified time, with the quality that was specified, of course you are going to get upset. The same goes for the customer, he or she might get upset, but if you notify them of the status of the release; it might be easier for them to accept that it is late or contain less functionality than specified, if you give a good reason for why this is the case.
11 Summary

- Keep the code clean and neat according to stated standards.
- Keep the code well documented, tested and commented during the entire development process.
- Make sure your code is reviewed thoroughly before releasing it to the customer.
- Make sure you keep the repository clean at all times and tag the versions of the software correctly.
- And the most important. *Think before you code!*
12 Recognitions
We would like to thank Daniel Lindh at Fareoffice for providing a lot of the information for the code review section. When developing the list he was inspired by the standards from OWASP (http://www.owasp.org) when producing this list.
Checklist for Software Development

1. Open task in Sharepoint
2. Solution design
3. Coding and development
4. Unit testing
5. Code commenting
6. Manual function test (functionality, validations etc)
7. Possible bug fixes
8. Code review (by yourself)
9. Possible code clean up
10. Documentation
11. Upload to test server and/or commit to SVN
12. Close task in Sharepoint
13. Common review at Sprint release (code review by supervisors)

Note: 3, 4, 5 should be conducted in parallel
A.2 Interview Protocols

A.2.1 Interviews with Erik Johansson and Tom Bergström

Follows on next page.
Interview regarding problem areas in the development in Indpro

This protocol is a summary of the answers to the questions in the interview. The interview has also been translated from Swedish to English.

**Interviewer [R]:** Ronny Roos

**Interviewee [E]:** Erik Johansson (Senior Technical Project Leader)

R: What is the main problem area in Indpro?

There are many problem areas but I feel one of the problems we have to struggle with most right now is a common problem for many outsourcing companies (not only offshore companies). The problem is to involve the customer in the project so that we get continuous feedback during the whole span of the project. Until now a large portion of Indpro’s customers has been entrepreneurs with none or very little technical skills which has been a large problem when providing specifications for a product or just understanding the development process. In projects where the customer is a technical person we experience much less problems in that area. We have also started using SCRUM as means of involving the client more into the development process that very clearly demonstrates different roles and responsibilities in the project. This has helped our customers understand what their role is in the project and made it much easier for them to provide the necessary information needed for development.

Time estimation and deadlines has also been a problem to us.

R: What do you mean about the time issue?

E: In general, time is not that important in India, if it takes two minutes or ten for the waitress to bring you your food doesn’t matter. The developers in Indpro often have the same mindset, “it doesn’t matter if the release is today or tomorrow, as long as it gets done”.

Time estimation has been a problem to us both in the sense that the developers have been reluctant to make time estimation and the fact that there has been little trust for the time estimation from a management perspective. This has resulted in that the management often used to force deadlines onto the developers in order to speed up the development process. Here SCRUM has also served Indpro well, by breaking down work into more manageable pieces it is easier for the developers to estimate time and they also grow more confident in the “art” of estimating. According to SCRUM the management is forbidden to interfere with the estimation process which has helped both developers to give more accurate estimation that they feel comfortable with and the management does not have to stress deadlines any more.

R: What would you say is the biggest difference between a Swedish and an Indian developer?

E: A Swedish developer often has more experience when he is fresh from school, and he isn’t only an expert in just programming, but has other knowledge. Like how to run a project, what is important to think of when designing software etc.
I can of course only speak from my personal experience in this matter. My belief is that a Swedish developer in general was exposed to computers at a very early age thus developing a very good understanding of computers, networks and software whilst his Indian counterpart might not have been in contact with computers until university or college level. This is an observation mainly regarding developers fresh out of school, when given the right guidance Indian developers are very fast to catch up.

*R: Would you say that this causes any problem for Indpro?*

E: Not direct problems. What is important to keep in mind is that we hire these developers consciously. If we wanted a developer with the same competence as a Swedish engineer, we would have to pay a lot more. What happens is just that more demands are put on us and the customers to make sure the developers work towards the correct goals.

*R: If we look at the development process itself, what do you have in place to ensure quality?*

E: In some project we have a complete development and testing process set up. But nothing is set up by us; the customer does all of that. We use SCRUM as our internal management and development process and if the client wants to use that as well then work out the parameters of the SCRUM together thus customizing the process after the clients and Indpro’s needs. Indpro can often contribute a great deal to the client’s development process given their experience in the offshore development market.

**End of interview**
Interview regarding problem areas in the development in Indpro
This protocol is a summary of the answers to the questions in the interview. The interview has also been translated from Swedish to English.

Interviewer [R]: Ronny Roos

Interviewee [T]: Tom Bergström (Process Manager)

R: What is the main problem area in Indpro?

T: The main problem is that there is a very big cultural difference between Swedes and Indians. A programmer from India generally doesn't think that much about design and cognition when designing a GUI for example and the usability of the system. They are also more concerned to “get it to work” functionality wise than spending time on maintainability and reusability of the code. Swedish customers often forget that this is the case and may not specify the exact rules for how to construct things in the applications.

R: Why is there no knowledge among the programmers regarding cognitive design?

T: There is a big difference in the education level between a software engineer from Sweden and one from India. In general, the first time an Indian developer uses a computer is when he gets to the university, in Sweden you are generally interested in computers and perhaps even programming when you start at the university. Furthermore the educations in India are skewed towards more theory based learning not involving so much “hands-on” practical exercises. This is completely opposite compared to Swedish educations that already from 1st grade in general are designed to promote problem solving skills.

R: Why is this the case?

T: Well, in general computers are far from common in the average home, meaning that most people don’t get access to computers before they get access to them in school or at work. In that sense Swedish programmers almost have a 15 year head start of practical experience. The reason that many people still decide to become developers is that as a developer you earn a lot of money compared to other professions. This makes it a highly respected profession which earns a high social status.

R: What would you say is the biggest difference between a Swedish and an Indian programmer?

T: In general the Swedish programmer has a much larger computer experience than the Indian programmer has. And of course as mentioned earlier a deeper rooted understanding of system usability and code quality.

R: What do you mean by that?

T: Well, as I said earlier the Indian programmers most often haven’t been around computers that much before they come to the university to start studying software engineering. This in turn means
that they most often don’t have the basic understanding about computers that the Swedish developers have.

*R: What problems would you say this causes for Indpro?*

T: The developers need more help with their systems, they can not maintain their systems in the same way a Swedish developer can. For an example, they bloat their desktops with unnecessary content, install a lot of unnecessary programs and last but not least most of them can’t fix basic problems with their computer by themselves.

This will of course mean increased system maintenance for Indpro and also increased time when developing applications that interact closely with the computer itself.

The level of bugs (both UI and logical) in the developed applications are higher than they should be due to the lack of knowledge in usability, maintainability and reusability.

*R: If we look at the development process itself, what do you have in place to ensure quality?*

T: Basically, this varies from project to project. There is no common development process, and almost everything is decided by the customers and they don’t always know what to include to ensure high quality. We ourselves include manual testing and code reviews in the development to try and minimize bugs and increase the quality.

We have observed improvements though; remember we are still a very young and new company, from the developers that have been with us for a longer time. They show a considerable higher quality in their code since they have experience working towards our Swedish clients. This means they have been forced to learn and take larger responsibility for conducting unit test, documenting their code and follow given standards and conventions.

**End of interview**
A.2.2 Summary of interviews to analyse implementation result
**Interview – Solution Evaluation**
Sprint 1 interview (2008-06-20).

This protocol is a summary of the answers to the questions in the interview.

**Interviewer [R]:** Ronny Roos

**Interviewees [D]:** Kiran Kalaskar, Prakash Kumar

**R:** What changes have occurred in your development environment?

D: So far, nothing has changed. We are still studying the process. We have installed the test tool and tried it out.

**R:** What changes have you made in the way you develop code?

D: In this sprint there has been extensive R&D, so we have not developed any code yet. Hence no changes have been made.

**R:** How has the quality of the code you develop changed?

Question not asked.

**R:** Do you feel that there are any changes in communication compared to before?

D: Communication is good as it is now, but we would like to have more contact with the customer. He is very hard to get hold of.

**R:** How do you know what the software shall fulfill?

D: This project is simply a migration project, so we can compare the functionality we develop with the original software.

**R:** How do you verify that the software fulfills its purpose?

D: We just compare it to the original software. In the end, the customer verifies that it handles the tasks correctly.

**R:** How do you feel that TestMatrix has affected testing of the code?

D: It simplifies the testing. You don’t have to switch development environment. The menus in TestMatrix are helpful and makes it easier to execute the tests than for an example standard NUnit.

**R:** How do you feel that the way of developing tests and code has affected your view of problem solving?

Question not asked.
R: How much of the suggested development solution would you say is implemented in your way of working right now?

D: Approximately 10-20% of the suggested solution is in place. Since we have not had a chance to do any actual development yet, we have not been able to put much of the solution in place.

End of interview
Interview – Solution Evaluation
Sprint 2 interview (2008-07-04).

This protocol is a summary of the answers to the questions in the interview.

Interviewer [R]: Ronny Roos

Interviewees [D]: Kiran Kalaskar, Prakash Kumar

R: What changes have occurred in your development environment?

D: We are now working more structured, we are following the development process which we now have on paper. There are different stages in completing a task in the way we work now.

R: What changes have you made in the way you develop code?

D: We are now following the suggested naming standards and also entering documentation comments in the code. In general the way we develop code is now more iterative than before.

R: How has the quality of the code you develop changed?

It has been raised. We are now both documenting the code when we develop it, as well as trying to think about performance issues. This was often overlooked without a formalized process.

R: Do you feel that there are any changes in communication compared to before?

D: The communication within the team is really good, but we have more or less no communication with the actual customer.

R: How do you know what the software shall fulfill?

Question not asked (same situation as last interview).

R: How do you verify that the software fulfills its purpose?

Question not asked (same situation as last interview).

R: How do you feel that TestMatrix has affected testing of the code?

D: We have planned to perform testing in a later stage in this sprint, so we have not gotten to know TestMatrix better than the last interview.

R: How do you feel that the way of developing tests and code has affected your view of problem solving?

Question not asked.
R: How much of the suggested development solution would you say is implemented in your way of working right now?

D: 50-60%. We have not yet inserted the testing into the process. We are planning to perform that in a later stage in this sprint.

End of interview
Interview – Solution Evaluation

This protocol is a summary of the answers to the questions in the interview.

Interviewer [R]: Ronny Roos

Interviewees [D]: Kiran Kalaskar, Prakash Kumar

R: What changes have occurred in your development environment?

D: More or less the last item in the development process has been added. That is, unit testing. However, we can’t follow the process entirely here, since we are only reconstructing small parts of the code and are very dependent on the third party control.

R: What changes have you made in the way you develop code?

D: Basically we are working in the same way as at the last interview. The only addition we have done is to create unit tests for the parts of the code that we are able to do so when we work with them.

R: How has the quality of the code you develop changed?

We can’t really say that the actual quality of the code we develop has changed. Much of the code we develop is tightly tied to the user interface and can’t be properly tested so we can’t say that the code we develop now is of higher quality. On the other hand, code that was previously developed, such as custom collections, now has unit tests, and we can be sure that they are working properly.

R: Do you feel that there are any changes in communication compared to before?

Question not asked (same situation as last interview).

R: How do you know what the software shall fulfill?

Question not asked (same situation as last interview).

R: How do you verify that the software fulfills its purpose?

Question not asked (same situation as last interview).

R: How do you feel that TestMatrix has affected testing of the code?

D: Compared to other test tools that we have worked with before, TestMatrix have several pros. It is embedded into Visual Studio and gives you data such as coverage built in into the tool. One of the biggest advantages is that we can easily create test classes, we do not have to create new projects and export them to dll-files to be able to test them.
**R: How do you feel that the way of developing tests and code has affected your view of problem solving?**

The way we solve problems has not really changed. What we have changed is the way we view the results of a problem that is solved. We can now be sure that all the code that is tested actually works according to the tests written and that we have not broken anything when we added new functionality.

**R: How much of the suggested development solution would you say is implemented in your way of working right now?**

D: About 90%. More or less everything is in place, but activities such as testing can’t be performed 100% according to the specified problems due to the fact that much of the code is already developed and tightly tied to the user interface due to the previous implementation.

**End of interview**
Appendix B

Program Listing

B.1 FibonacciGenerator.cs

Follows on next page.
using System;
using System.Collections.Generic;
using System.Text;

namespace TestExampleProject
{
    class FibonacciGenerator
    {

        public LinkedList<int> generateFib(int to)
        {
            LinkedList<int> fibList = new LinkedList<int>();
            fibList.AddLast(0);
            if (to > 0)
            {
                //We're safe
                fibList.AddLast(1);
                int lastItem;
                int secondLastItem;
                while (to > fibList.Last.Value)
                {
                    lastItem = fibList.Last.Value;
                    fibList.RemoveLast();
                    secondLastItem = fibList.Last.Value;
                    fibList.AddLast(lastItem);
                    fibList.AddLast(lastItem + secondLastItem);
                }
            }
            return fibList;
        }
    }
}
B.2  Automatically Generated Tests by .TEST

Follows on next page.
using System;
using NUnit.Framework;
using Dottest.Framework;
using Dottest.Framework.RecordState;

namespace TestTestExampleProject
{
    [TestFixtureAttribute()]
    [TestFixtureContext("TestExampleProject.FibonacciGenerator")]
    public class TestFibonacciGenerator
    {
        [SetUpAttribute()]
        public void SetUp()
        {
            // TODO:  Add SetUp() implementation
        }

        [TearDownAttribute()]
        public void TearDown()
        {
            // TODO:  Add TearDown() implementation
        }

        [TestCaseAutogenerated()]
        [TestCaseAuthor("Ronny")]
        [TestCaseContext("TestExampleProject.FibonacciGenerator.generateFib(System.Int32")]
        [ExpectedException("System.ArgumentException")]
        [TestCaseUnverified("Test case not verified")]
        [TestAttribute()]
        public void TestGenerateFib_0()
        {
            object inputThis = Helper.CreateObject("TestExampleProject.FibonacciGenerator");
            int inputArg0 = 0;
            System.Collections.Generic.LinkedList<System.Int32> var0 =

            #region Record State
            vr.ObjectValue(inputThis, "TestExampleProject.FibonacciGenerator", "inputThis");
            vr.Int32Value(inputArg0, "inputArg0");
            vr.FinishRecording();
            #endregion
            // State recording could not complete because
            // the test threw a System.ArgumentException
        }

        [TestCaseAutogenerated()]
        [TestCaseAuthor("Ronny")]
        [TestCaseContext("TestExampleProject.FibonacciGenerator.generateFib(System.Int32")]
        [TestCaseUnverified("Test case not verified")]
        [TestAttribute()]
        public void TestGenerateFib_1()
        {
            object inputThis = Helper.CreateObject("TestExampleProject.FibonacciGenerator");
            int inputArg0 = -1;
            System.Collections.Generic.LinkedList<System.Int32> var0 =

            #region Record State
            vr.ObjectValue(inputThis, "TestExampleProject.FibonacciGenerator", "inputThis");
            vr.Int32Value(inputArg0, "inputArg0");
            vr.FinishRecording();
            #endregion
            // State recording could not complete because
            // the test threw a System.ArgumentException
        }
    }
}

// This code was generated by Parasoft .TEST.
vr.ObjectValue(inputThis, "TestExampleProject.FibonacciGenerator", "inputThis");
vr.Int32Value(inputArg0, "inputArg0");
vr.FinishRecording();
#endregion
#region Assertions
Assert.AreEqual(-1, inputArg0);
#endregion Assertions for var0
Assert.IsNull(var0.Last.Next, "Next should be null");
Assert.AreEqual(0, var0.Last.Value);
Assert.IsNull(var0.Last.Previous, "Previous should be null");
Assert.AreEqual(1, var0.Last.List.Count);
Assert.IsNull(var0.First.Next, "Next should be null");
Assert.AreEqual(0, var0.First.Value);
Assert.IsNull(var0.First.Previous, "Previous should be null");
Assert.AreEqual(1, var0.First.List.Count);
Assert.AreEqual(1, var0.Count);
#endregion
#endregion

[TestCaseAutogenerated()]
[TestCaseAuthor("Ronny")]  
[TestCaseContext("TestExampleProject.FibonacciGenerator.generateFib(System.Int32)")]
[TestCaseUnverified("Test case not verified")]
[TestAttribute()]  
public void TestGenerateFib_2()  
{
    object inputThis = Helper.CreateObject("TestExampleProject.FibonacciGenerator");
    int inputArg0 = System.Int32.MinValue;
    
    #region Record State
    vr.ObjectValue(inputThis, "TestExampleProject.FibonacciGenerator", "inputThis");
    vr.Int32Value(inputArg0, "inputArg0");
    vr.FinishRecording();
    #endregion
    #region Assertions
    Assert.AreEqual(-2147483648, inputArg0);
    #endregion Assertions for var0
    Assert.IsNull(var0.Last.Next, "Next should be null");
    Assert.AreEqual(0, var0.Last.Value);
    Assert.IsNull(var0.Last.Previous, "Previous should be null");
    Assert.AreEqual(1, var0.Last.List.Count);
    Assert.IsNull(var0.First.Next, "Next should be null");
    Assert.AreEqual(0, var0.First.Value);
    Assert.IsNull(var0.First.Previous, "Previous should be null");
    Assert.AreEqual(1, var0.First.List.Count);
    Assert.AreEqual(1, var0.Count);
    #endregion
    #endregion
    
    [TestCaseAutogenerated()]
    [TestCaseAuthor("Ronny")]  
    [TestCaseContext("TestExampleProject.FibonacciGenerator.generateFib(System.Int32)")]
    [TestCaseUnverified("Test case not verified")]
    [TestAttribute()]  
    public void TestGenerateFib_3()  
    {
        object inputThis = Helper.CreateObject("TestExampleProject.FibonacciGenerator");
        int inputArg0 = 1;
    }
System.Collections.Generic.LinkedList<System.Int32> var0 =
new object[] { inputArg0 }, new System.Type[] { typeof(System.Int32) })};

#region Record State
vr.ObjectValue(inputThis, "TestExampleProject.FibonacciGenerator", "inputThis");
vr.Int32Value(inputArg0, "inputArg0");
vr.FinishRecording();
#endregion

#region Assertions
Assert.AreEqual(1, inputArg0);
#endregion Assertions for var0
Assert.IsNull(var0.Last.Next, "Next should be null");
Assert.AreEqual(1, var0.Last.Value);
Assert.AreEqual(0, var0.Last.Previous.Value);
Assert.AreEqual(2, var0.Last.List.Count);
Assert.AreEqual(1, var0.First.Next.Value);
Assert.AreEqual(0, var0.First.Value);
Assert.IsNull(var0.First.Previous, "Previous should be null");
Assert.AreEqual(2, var0.First.List.Count);
Assert.AreEqual(2, var0.Count);
#endregion

#region
[TestCaseAutogenerated()]
[TestCaseAuthor("Ronny")]
[TestCaseContext("TestExampleProject.FibonacciGenerator.generateFib(System.Int32)")]
[TestCaseUnverified("Test case not verified")]
[TestMethod()]
public void TestGenerateFib_4()
{
    object inputThis = Helper.CreateObject("TestExampleProject.FibonacciGenerator");
    int inputArg0 = System.Int32.MaxValue;
    System.Collections.Generic.LinkedList<System.Int32> var0 =
new object[] { inputArg0 }, new System.Type[] { typeof(System.Int32) })};

    #region Record State
    vr.ObjectValue(inputThis, "TestExampleProject.FibonacciGenerator", "inputThis");
    vr.Int32Value(inputArg0, "inputArg0");
    vr.FinishRecording();
    #endregion
}

#endregion

#region
[TestCaseAutogenerated()]
[TestCaseAuthor("Ronny")]
[TestCaseContext("TestExampleProject.FibonacciGenerator.generateFib(System.Int32)")]
[ExpectedException("System.ArgumentException")]
[TestCaseUnverified("Test case not verified")]
[TestMethod()]
public void TestGenerateFib_5()
{
    object inputThis = Helper.CreateObject("TestExampleProject.FibonacciGenerator");
    int inputArg0 = 2;
    System.Collections.Generic.LinkedList<System.Int32> var0 =
new object[] { inputArg0 }, new System.Type[] { typeof(System.Int32) })};

    #region Record State
    vr.ObjectValue(inputThis, "TestExampleProject.FibonacciGenerator", "inputThis");
    vr.Int32Value(inputArg0, "inputArg0");
    vr.FinishRecording();
    #endregion
}

#endregion
vr.ObjectValue(inputThis, "TestExampleProject.FibonacciGenerator", "inputThis");
vr.Int32Value(inputArg0, "inputArg0");
vr.FinishRecording();
#region
    // State recording could not complete because
    // the test threw a System.ArgumentException

/** Do Not Edit: Hashcode[1586be40] */
}