

Making Architectural Design Phase Obsolete – TDD as a Design Method

Marc Josefsson
T-76.650 Seminar course on
SQA in Agile Software Development
Helsinki University of Technology
Finland
Mjosefss@cc.hut.fi

Abstract— The architecture of a software system is one of its most valuable attributes. Good architecture supports software testing, reusing, maintenance and adaptability to changes. The ways to create architectural design may vary, but its role is unquestionable.

The advocates of agile software development methods are claiming that software projects can be successful without a separate architectural design phase. One of the supporting methods of agile methodologies is Test Driven Development (TDD). What is TDD's impact on the quality of software design? Is it possible to create a solid architectural base by applying TDD in practice? This article tries to enlighten the role of TDD while creating high quality architectural design.

Index Terms— Test Driven Development, TDD, TFD, Agile software design.

1. INTRODUCTION

Test Driven Development (TDD) (Beck, 2003) has gained much attention during the last few years. The reason for the increasing interest has been the rise of eXtreme Programming (XP) (Beck, 2000; Auer and Miller 2001). TDD is a fundamental part of XP's methodology, since all other elements of XP are built on unit testing and test first development. The popularity of TDD has also given the methodology a lot of different names: Test First Design (TFD), Test First Programming (TFP) and Test Driven Design (TDD). TDD as a practice highlights writing unit tests before the actual code implementation. The written unit tests describe the needed interfaces between different software components or classes. Due to this fact all coding should be done after thinking of the current method's or object's needs. This process model should minimize the time between decision points and feedback as the developer gets instant feedback on his design during the implementation of his recent design.

The advocates of TDD are speaking of high increase in the software quality (both quality of design and quality of code e.g. passing the designed test cases). For example Kent Beck asserts (Beck, 2001), "Test-first code tends to be more cohesive and less coupled than code in which testing isn't a part of the intimate coding cycle". The continuous refactoring and test first point of view are said to decrease the need for "big design up front", which is also known as BDF (Beck, 2000). This clause is claimed to be very fundamental and critical for the success of software projects. Most of the

software projects are balancing between the need of stability and guidance from design process, and between the need to adapt to changes. Still most of the recent studies on TDD's impact on software's quality have concentrated on TDD's effects on the quality of the code and/or usage of resources during the development process.

Maintainability of the developed software is a big issue if a company wants to manage the competition in the market. Creating software products increases the need for reusable software components. Maintaining and tailoring the software must be easy in order to create as much revenues as possible. In order to create as maintainable software components as possible, the quality of the architectural design is critical (Sommerville, 2001).

This study concentrates on the fact that the advocates of TDD are saying that there is no need for separate design phase before the implementation phase. However the quality of architectural design is clearly seen as one of the greatest reasons for a software project's success or failure (MacCormack, 2001). This study tries to find answer to the following question: Does TDD help to create software of good enough quality when it comes to the quality of design? This study tries to identify the size of the impact TDD has on the quality of design, and what are the differences between successful projects and projects that have turned out to be failures. This study will be conducted as a literal study. Since the field of agile software development is quite new most of the papers included in this study will be written during or after the year 2003.

The rest of the paper is structured as follows: In chapter 2 TDD is taken under more precise inspection. This chapter describes TDD as a methodology in general. After the general overview the chapter lists all the proposed impacts that TDD should have on the quality of software design. Chapter 3 lists the results of the study. The chapter is split into subchapters, which list both cases supporting the usage of TDD and cases against the usage of TDD. The limitations of each study are summarized after the study. Chapter 4 is reserved for discussion about the found facts. The study's conclusions are presented in chapter 5

2. OVERVIEW ON TDD

2.1. General overview

TDD as a practice deals with the implementation phase of the software. The main reason for adapting test driven development is usually to eliminate faults as early as possible. After the faults have propagated to the operational phase of the system the removal costs of the faults will be substantially higher than when fixing the faults earlier (Boehm, 1981). TDD is about short iterations which include the following phases:

- Create a small set of automated unit test cases, which are enough to cover the task under development.
- All the written test cases should fail, because of the fact that there is no code to execute yet.
- Write the actual implementation which follows the rules set by the unit tests designed before (the goal is to write code which makes the unit tests to pass).
- Rerun the tests to ensure that all the tests pass.
- Regression testing is used on a regular basis to ensure that the current changes to the system have not broken anything that was implemented before.

This iterative process is used to develop a small piece of new functionality defined in the requirements. Generally the effort required to complete a task should not exceed one day's work. Due to the fact that all the unit tests must run correctly after the newly implemented code is added to the repository, there can be some confidence that the current version of the system works correctly. The developers are entrusted with the responsibility of unit test design. This clearly means that the developers must have some experience on the test case design process (the different exceptions and branches should be handled with the unit test cases).

XP and some other methodologies state that there is no need for upfront design (Beck, 1999). In XP there is not any formal design before the actual implementation phase. The need for upfront design in TDD has been researched by Bob George (George, 2002). He came to the conclusion that TDD can and should follow a process where no separate design phase is needed. But it should be stated that TDD is an independent process: TDD is flexible and it can be used with or without an upfront design phase. TDD as such tries to highlight the importance of creating unit tests before implementing the actual code rather than writing test cases after the actual code. But in the recent past TDD has been linked with agile methodologies such as XP. That is why TDD's need for upfront design phase should be researched.

The guideline to architectural design is simple if you listen for example the developers of XP. By using constant refactoring and general metaphor to describe the overall system structure one should be able to produce the simplest design which fulfills the given requirements. If a change in design needs to be made, then the change is implemented in the software. Simple design is the cornerstone of XP: one should never spend too much time on creating a complex design for a system, if the need for such a system does not exist.

2.2. Proposed impact on design

The inventors and advocates of TDD tell about great improvements in design's quality when TDD is taken as process guideline (Williams, 2003). Simple design should produce code with high cohesion and low coupling. By using only the simplest possible design one should be able to adapt to all the occurring changes in the requirements. This sounds intriguing. Most of the published studies or researches have been conducted on the effects TDD has during the software implementation phase. But these studies should also take into account the full life span of the product and to measure the maintainability of the software during its life cycle. That is because the quality of design stands out when great changes or maintenance tasks are about to happen. For example, to add new modules to the system might be impossible (or really hard) if the interfaces of the system are not clear and if the responsibilities of different components are not clear and simple enough.

The central idea that sticks out when looking TDD from the eyes of a traditional software developer is that TDD states that the design process during the implementation phase is adequate to produce high quality software.

3. RESULTS OF THE STUDY

3.1. Result overview

The greatest problem while collecting the material to this study was to find enough material where the design aspect of TDD would have been considered. It seems that there is call for wide enough studies on TDD's impact on the quality of design. The following chapters list cases where TDD has proven to be successful or unsuccessful, respectively. I will summarize the basic information and results of every study in the beginning of each of the chapters. After that I will bring out the limitations of the studies. The comparison between the pros and cons will be made in chapter 4. Discussion.

3.2. Cases where TDD has improved the quality of the design

The interest towards TDD has increased during the last few years. But still the number of experiments on TDD is low. I had some problems trying to find objective or otherwise valid scientific material on TDD's favor. That is why I have also included studies that are not very significant scientifically (studies conducted with university students and/or with low number of participants). This chapter lists successful cases where TDD has been used as one of its main practices in development. Table 1 summarizes the chapter.

Muller and Hagner conducted an experiment trying to compare TDD with traditional programming using development time, resultant code quality and understandability as the terms of the study (Muller, 2002). The experiment conducted with 19 graduate students. The subjects were split into two groups, TDD and control group. Both of the groups were supposed to solve the same tasks based on the given design and specification documents and method declarations.

The TDD group followed the methodology, and wrote every

	Studies			
	Muller and Hagner (2002)	George (2003)	Maximilien (2003), Williams (2003)	Rasmusson (2003)
Number of participants	19	24	9	13
Environment	University	Commercial	Commercial	Commercial
Main focus of the study	Quality (number of defects) and resource usage	Quality (number of defects) and resource usage	Quality (number of defects) and resource usage	General
Type of results when it comes to the quality of design (qualitative/quantitative)	Qualitative	Qualitative	Qualitative	Qualitative
How the design process was handled	Students received full design documents in the start of the project	Should have followed TDD, but faced major slippages from test first development	All the design was made up-front	According to TDD
Summary of the results (from the point of view of design process)	Understandability of the program created with TDD increased: the existing interfaces were reused properly	The developers felt that TDD facilitates simpler design and that lack of upfront design is not a hindrance	The participants of the project felt that TDD had great impact on the quality of the product's design.	TDD worked well in the project. TDD helped the developers to stay in the simplest possible design.

Table 1 Details of successful projects that used TDD.

test case before the actual implementation. The project had two phases to model modern software projects. The project started with an implementation phase, after which the groups were informed about their test passing rate and failed test cases. An acceptance test phase followed the first phase, in order to give the groups a possibility to fix the bugs in their code.

Based on the results acquired during the experiment, Muller and Hagner concluded that test-first programming does not increase the quality of the software or decrease the development time. The benefits of TDD were clearly seen in the understandability of the program, since the interfaces of the implemented components were properly reused while using TDD.

There were quite many limitations in this study. Both the number of participants and the participants' experience on TDD was quite low. This study is a good example of the interests of the researchers in the field of software engineering: only things that are considered quantitatively are the number of defects and the usage of resources. TDD's proposed impact on the quality of design is purely qualitative and the knowledge was acquired during an informal interview. Since the participants of this study received the design documents and the body of the necessary methods beforehand, the study failed to measure the quality of design extensively enough.

Another experiment (George, 2003) was run with 24 professional programmers in commercial environment. Three companies were involved (John Deere, RoleModel Software and Ericsson) with one eight person group per company. The experiment was structured on company level as follows: one half of the developers adapted TDD as their development methodology, while the other half stayed with waterfall-like approach. Both groups had to practice pair programming, and

they were to develop a quite simple bowling game satisfying the given requirements.

The effectiveness of TDD was measured using two metrics: the development time; and the results of the functional testing. Also the quality of the unit tests written by the TDD groups was measured using code coverage analysis.

Based on the results of the experiment TDD appears to produce higher quality code. The code produced with TDD passed more test cases than the code produced with traditional practice. The results also showed that the developers using TDD took a little while longer developing their code (16%), but their code passed 18% more tests than the other group. This study produced only qualitative information on TDD's impact on the quality of design. The developers felt that TDD facilitates simpler design and that the lack of upfront design was not a hindrance.

The problem of this experiment was that the TDD group did not follow the given instructions by the book, resulting in only a few written test cases. There existed also other reasons, which introduced some limitations on the external validity of the experiment. The main limitation was the small size of the produced application, since the typical size of the software was only 200 LOC. Also the developers' experience level on TDD varied a lot between the sites, which makes the comparison between the results difficult.

One more experiment took place at IBM (Maximilien, 2003; Williams, 2003). They had a legacy product which had gone through seven releases since late 1998. The experiment compared the implementation process of the legacy product and the implementation of similar new product, which had less features, and worked on less platforms than the legacy product. The experiment's TDD group consisted of 9 professional programmers. Some of the group members were not familiar with TDD as a development practice when they

	Studies		
	Theunissen (2003)	Müller and Padberg (2003)	MacCormack et al. (2001)
Number of sample projects	-	-	29 projects
Environment	Academic	Academic	Commercial
Main focus of the study	How agile methods conform to standards like ISO 12207:1995	Economic evaluation of XP projects	Characteristics of an effective development process
Type of results when it comes to the quality of design (qualitative/ quantitative)	Qualitative things that one has to do in order to conform to different standards while using agile methods.	Qualitative	Quantitative and qualitative
Summary of the results (from the point of view of design process)	Metaphor by itself is not enough to facilitate the architectural design of the system. One must adapt XP to the needs of the environment. This means that one must develop an agile way to create architectural design documents.	The absence of architectural design documents can be an economic risk. The customer must be able to be completely sure that the produced software really conforms to the given requirements. This can be difficult if any documents do not exist.	Well done architectural design is one of the most important attributes of successful software projects.

Table 2 Details of experiments against the usage of TDD.

started the project.

The nature of the project was to create new code, while the legacy product project just maintained existing code. The developers decided to design the whole product upfront, since the requirements of the product were to be stable. The ready design documents described all the necessary method and class interfaces that the product would need. After the design phase the actual implementation process and usage of TDD begun. To guarantee that all unit test cases would be run by all members of the team, the build process was automated and run daily both locally and remotely.

The results of the experiment were as following: the defect density rate was 50% lower than the developers using traditional development practices. The impact on developers' productivity was minimal. The developers felt that TDD had a great impact on the design of the product: they could go through some major changes in the requirements two-thirds of the way. The created product was found testable and new components could be easily integrated into the system. The size of the impact on design is based only on the feelings of the developers. This experiment had the same problem as the preceding ones when TDD's impact on design process is considered: there is no metrics or other relevant data about the quality of design. The limitations of this experiment are the small number of participants and the type of results collected (qualitative).

ThoughtWorks used XP in one of their most successful projects ever (Rasmusson, 2003). The actual group of developers comprised 13 people with varying experience level. Upon delivery, the project code base had 21000 lines of application code and 16000 lines of test code. The project team felt that one of TDD's advantages was that while doing

the test cases, and thus creating the design of the product the design process was intuitive and easy to follow. The project was all in all a mixture of different practices, since this was the first XP project for most of the engineers that participated in the project: all the methods used in XP require customization and time to adapt. The project group felt that adapting XP metaphor helped them in creating better insight of the system.

The project personnel used agile methods when needed. For example it was estimated that about one-third of the overall implementation effort was done according to TDD. The process of the project was all in all very adaptable: the best way of doing things was chosen on the fly rather than trying to adjust the whole process to use some certain process model.

In this project all the introduced data is in qualitative form only. No actual metrics supporting the author's words are available. But still this article gives additional information on the usage of TDD. The undergone project had many typical features of software projects: the environment of the project was very unstable and the adaptation of new process model was slow. The new methodology chosen for the company (XP) was adapted to the project's needs very well: the personnel were allowed to deviate from the process model when appropriate. This resulted in a project where every employee was highly committed and used all the suitable techniques available to produce a high quality product.

3.3. Studies against the usage of TDD

The critics of XP and TDD are claiming that the need for architectural (upfront) design is a fact. The most common argument used against TDD is the fact that radical changes in the software's future would cause great costs (Mugridge,

2003). Some upfront design is needed to reduce the risks of critical changes. Can the quality of iterative design be high enough to handle changes in the requirements also in the late phase of the project? Table 2 summarizes the found results of this chapter.

Based on research of how agile methods adapt to different standards it can be said that XP on its own is not enough to handle the requirements of what is thought to be a stable development process (Theunissen, 2003). But as the name "agile methodologies" state, the methods are able to conform to dynamic environment. With agile methods one can also support a software process, which conforms to international standards like ISO 12207:1995. But some changes in the methodologies are needed. For example, XP metaphor is not enough to facilitate the architectural design of the system. Some additional information must be collected from planning game and white-board discussions. After that the project members are able to create better vision about the architectural structure of the system, in other words create an architectural design.

The way that the architectural design is created can be made agile. But in order to maintain the agile nature of the process, the architectural design document creation should be made automatic. The existence of up-to-date architectural design documents are thought to be a sign of a mature software process. The problem with agile methodologies is that the experience gathered by the project members during the project is not published to other employees of the company. This is a direct consequence of the absence of adequate documentary. This study is not actually against the usage of TDD in general. It just states that one should not use TDD as it is if a stable development process is to be created.

The absence of design and requirement documents can also be an economic hindrance (Muller and Padberg, 2003). If the creation and updating of proper documentation is neglected, it might be difficult to guarantee that the software actually fulfills the needs of the customer. Based on the results of the study, the economic advance that might be achieved by using agile methodologies depends greatly on the speed and quality advantage. While the actual advantage might remain quite small, the software should conform to customer requirements very well. This can be hard to verify, if no actual structure of the designed product is at hand. The value of architectural design document together with other documentation is not only in stable development process, but also in validation of the produced software.

The heaviest proof against TDD is presented in a wide research on the field of software development (MacCormack et al., 2001). The research material consists of 29 completed projects from 17 companies. The research data of the project was collected with both by interviewing the project managers in the beginning of the study and by surveys sent to the project managers later during the study. The need for architectural design was a part of the study. The information about the need of architectural design was collected in the following way: The project managers were asked to break down the resources that they had devoted to four categories: project management,

architectural design, development and test. The study was then based on calculations made on these results. The studied projects types were very heterogeneous, which increases the value of this research. The main common factor among the software projects was that they all were implemented in a very dynamic environment. On the other hand the study supports the usage of agile methodologies. The metrics showed that the ability to get early feedback from customers is essential to the success of software projects. This is also supported in TDD, since the passed unit tests are the earliest possible feedback from the system.

The need for good architectural design arises from the fact that current market environment is usually very unstable. The products must be able to adapt to very large changes in the requirements also in the late stages of the project. This can only be achieved by building the product a high quality foundation. TDD advocates say that the way to adapt to changes is creating as simple design as possible, and refactoring the code as often as possible, thus making it clear and manageable. But MacCormack's research indicates that the best way to produce software design that can manage changes is to create good architectural base before starting the actual project. Sensible architecture is able to fulfill the customer needs, be cost effective and adapt to changes.

The research indicates that the customer requirements are not well known. This includes both the functional and non-functional requirements. Agile way to handle requirements is to have close interaction with the customer. This way the developers should be able to communicate with the customer, and to produce software that the customer really wants. The problem is that the base architecture of the software should be such that the developers are able to implement all the upcoming tasks: the software must adapt to changes that can cause even quite large changes in the software. The delivered software should contain overall architecture which best captures the customer's needs (performance, and other non-functional requirements).

One more architectural challenge is to handle the incompleteness of the product while serious integration is done. The architecture must allow changes to components where they are needed while other components remain unchanged or changes are at least minimized. When all the three preceding facts are combined (good architecture makes changes possible, good architecture allows integration of incomplete components, incomplete customer requirements in the beginning of the project) the following question rises: should every project use greater resources in the beginning of the project to create feasible software architecture?

The research indicates that the projects that dedicate greater resources to architectural design come up with higher quality products which can be integrated to existing systems earlier. Good design maximizes the end products performance, while keeping the actual development as flexible as possible. There exist facts that are limiting the trustworthiness of the results of this study. The researchers are claiming that all the effort used in architectural design is used in the beginning of the project. The setup of this study does not properly state the effort used

in architectural design during the project (agile way) and in the beginning of the project (traditional way).

4. DISCUSSION

In general it can be said that the impact that TDD has on the quality of design is based on qualitative results only. According to the material found on the internet, even the successful experiments did not contain any quantitative results on the need of architectural design phase. All the developers that participated in the projects presented in chapter 3.2 felt that the usage of TDD created simpler design than traditional methods. But these results were merely based on the feelings of the participants. If the projects led by the creators of XP are taken under examination (Beck, 1999), the need for solid experience base rises above everything else. One cannot create good architectural design on the fly, if he does not possess enough experience and knowledge about the field of the software.

In most of the projects that I have listed in this paper as the advocates of TDD, the design documents of the software were given to the developers in the beginning of the project. It is difficult to actually measure the quality of design created with TDD when most of the design is created for the developers. Two of the projects created their own design documents, but the other used separate design phase in the beginning of the project. The remaining project was not a pure agile project. They adapted different process models to their needs: the process model was chosen according to the needs of the task at hand. This project (Rasmusson, 2003) is a good example of software project using agile methodologies. Agility in software processes means that the process can adapt to the needs of the environment. But even in quite small projects the needs of the software system in different phases of the project can change. This means that even the process model must be able to change with the project. If the project members are motivated and experienced enough, one can create a product with high quality design (Rasmusson, 2003).

The opponents of TDD listed in this paper are claiming that the existence of architectural documents is essential for the success of software projects. These claims were viewed from the point of view of software standards (Theunissen, 2003), the economic of XP projects (Muller & Padberg, 2003) and an empirical study which examined the characteristics of 29 successful software projects. Based on these studies it seems that in most of software projects the existence of good architectural design is crucial to the success of the project. I think that these studies do not properly consider the way in which the architectural design is created. While the design process might be critical to the success of the software, the need for documents is not so straightforward. TDD could be used to create the architectural design of the system, but the design does not necessarily have to be documented.

Based on the studies that I have read during the creation process of this paper I think that most important attribute of project manager is to be able to choose the right process model for different environments. If the experience level and the knowledge base of the developers is too low, the project

should not use TDD-based process as their only process model. The success of agile software development is greatly based on the skills of the people involved in the project. The size of the project organization can also be a limitation while choosing the right process model for the project (Elssamadisy & Schalliol, 2002). That is because agile methodologies were originally designed to be used with smaller development teams.

The present studies (the ones that I have listed here and other studies on TDD) do not properly consider TDD's impact on the quality of design. It was really a problem trying to find enough material for this study. During the writing process of this paper I realized that the nature of this study is different that I had thought. I could not merely collect studies that would have something to do with the design process in TDD. Most of the articles written on agile methodologies purely list the claimed advantages of the process model. This also the case with TDD: the proposed impact on the quality of design is based only on assumptions and the feelings of the developers. Personally I would be really interested in an empirical study where the life cycle of the produced software is taken under consideration. The creation of software can be handled with other process models also, but if agile methods have the proposed impact on the quality of design the biggest effect will be in the maintenance phase of the software. Maintaining and reusing software produced with TDD should be easier than software produced with traditional methods. I would be interested in studies that measure the effort used to both study the existing code base and create new code (or reuse) based on the existing implementation.

The literature study that I conducted showed the absence of both studies concerning TDD's impact on architectural design before the actual implementation phase and the impact on quality of design in general. The only result of this study was the still ongoing contradiction between the advocates and the adversaries of agile methodologies: based on the results of this study the existence of solid architecture is essential for a software product. But this study could not find a solution to the problem that in which way it is best to create the architectural design. The answer might be as simple as: it depends. It depends both on the needs of the project and the characteristics of the project (experience level, amount of resources, schedule etc.).

There is a certain need of thorough empirical study, which would examine the need of upfront design in development process. Also an empirical study which would cover the whole life span of a software product would be needed: all current studies cover only the mature phase of the products. But the actual value of good design comes up only in maintenance phase or when trying to reuse the components. All the variables that have an effect on the success of a software project should be measured. These variables include the size of the project (in number of developers and/or lines of code), type of the project (how dynamic the requirements are etc.), experience level of the developers (TDD without any upfront design might either require great experience in the current field of the project, or great experience in usage of TDD as

such).

TDD as such does not require the usage of any specific process methodology. But TDD is mainly included in agile development methodologies. That is why the discussion about the need of architectural design has risen. The need for upfront design might be the greatest hindrance in using TDD in more complex projects.

5. CONCLUSIONS

This literature study covers four studies that are supporting the usage of TDD and which are claiming that the usage of TDD has resulted in higher quality code. Also three articles that are against the usage of TDD or agile methods in general have been introduced. The question whether the usage of TDD results in higher quality of produced design remains unanswered after this study. The studies that are supporting TDD are based on very limited experiment setups and therefore cannot be generalized in a sensible way. Also the papers opposing the usage of agile methods contain generalizations. But the main value of this study was to realize that still much research has to be done. The claimed effect that TDD should have on the quality of design cannot be proven right with the present studies.

The value of architectural design seems to be high, and both the advocates and the adversaries of agile methodologies approve that every piece of software needs a solid design. Based on the papers that I read through during this literature study I would say that the need for good architectural design exists. The real question is how to produce it. This study toughened up my opinions about different software process models: the real resource of success of software projects is within the project personnel. Project managers should not blindly follow any process model. One should always choose the best process model: the process model used should always support the implementation work done as well as possible, since the only goal of software engineering is to produce executable software. It is not about the means, but about achieving the goal.

6. REFERENCES

- Auer, K., and Miller, R., *XP Applied*, Addison Wesley, 2001
- Beck, K. *Extreme programming explained* (Reading, MA, 2000), Addison-Wesley, p.157.
- Beck, K. *Embracing Change with Extreme Programming*. IEEE Computer, 1999, vol. 32, no 10. Pp. 70-77.
- Beck, K., "Aim, fire", IEEE Software, 2001, vol. 18, no. 5, Pp. 87-89.
- Boehm, B. W., *Software Engineering Economics*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1981
- Elssamadisy, A., and G. Schalliol, *Recognizing and responding to "bad smells" in extreme programming*. Proceedings of the 24th International Conference on Software Engineering, 2002. Pp. 617-622.
- George, B., and Williams, L., *An Initial Investigation of Test-Driven development in Industry*, Proceedings of the ACM symposium on Applied computing, March 2003.
- George, B. *Analysis and Quantification of Test Driven Development Approach* MS Thesis, Computer Science, 2002.
- Jeffries, R.E., "Extreme Testing", *Software Testing and Quality Engineering* 1999
- Kaufmann, R., and Janzen, D., *Implications of Test-Driven Development: A Pilot Study*, ACM, 2003, Vol. 10, pp. 298-299
- MacCormack, A., et al., *Developing Products on "Internet Time": The Anatomy of a Flexible Development Process*, Management Science, 2001, Vol. 47, No. 1, pp. 133-150.
- Maximilien, .E, and Williams, L., *Assessing Test-Driven Development at IBM*, IEEE, 2003, pp. 564-569
- Mugridge, R., *Test Driven Development and the Scientific Method*, Proceedings of the Agile Development Conference (ADC '03), 2003.
- Muller, M., And Hagner, O., "Experiment about Test-first Programming", presented at Conference on Empirical Assessment in Software Engineering (EASE), 2002.
- Muller, M. And F., Padberg, *On the Economic Evaluation of XP Projects*, ACM, 2003, Vol. 9, pp. 168-177.
- OOPSLA '03, October 26-30 2003, Panel: Discipline and Practices of TDD, ACM, 2003, Vol 10., pp. 268-270.
- Rasala, R., *Embryonic Object versus Mature Object: Object-Oriented Style and Pedagogical Theme*, ACM, 2003, Vol. 6, pp. 89-93
- Rasmusson, J., *Introducing XP into Greenfield Projects: Lessons Learned*, IEEE Software, 2003 (May/ June), pp. 21-28
- Sommerville, I., *Software Engineering*, Addison Wesley, 6th edition, 2001
- Theunissen, W.H., et al., *Standards and Agile Software Development*, Proceedings of SAICSIT 2003, pp. 178-188.
- Williams, L., et al., *Test-driven development as a defect-reduction practice*. 14th international symposium on software reliability engineering, 2003, 17-20 Nov. Pp. 34-45.