Reviews/inspections and pair programming

Richard Berntsson Svensson, Jimmie Johansson, Kashif Ahmed Khan
School of Engineering
Blekinge Institute of Technology
dj20p@ny.com, jj76@spray.se,kakb04@student.bth.se

Abstract

When developing software there are lot of defects brought into software products. Many of the defects are discovered during the testing of the product. Still, many unknown defects are present but invisible. To decrease defects in the final product, defect prevention can be used to reduce the defects by preventing them from getting into the product in the first place. This report brings some understanding of defect prevention and techniques that can be used for preventing these defects. The conclusion is that many defects can be prevented from slipping into the code but also that reviews and inspections are not easy to perform. They are huge processes with both advantages and disadvantages. However, they might be well worth the effort if benefits can be proven.

1. Introduction

This paper is a part of an assignment in the course Quality Management given by Blekinge Institute of Technology. The purpose with the assignment is to present the concept of inspections and reviews and how they can prevent defects in software development. As the report is written with defect prevention in mind, the concept of pair programming is also discussed.

Defect prevention is proven very effective. Two thirds of the maintenance cost is reduced by defect prevention and five to ten times reduced cost from testing the product [4]. It is with these figures in mind we started to bring some understanding in the review techniques.

The report has the following outline. In section 2, a summary of inspections, walkthrough and pair programming is made. In section 3 we describe the defects that can be found prevented by these techniques. In section 4, a discussion about acceptability of the techniques is held. In section 5 we discuss advantages and disadvantages and in section 6, the relation of these techniques to ISO and CMMI. Finally, we conclude the report in section 7.

2. Summary of techniques

The purpose with reviews is to evaluate the process and the product to be able to find defects and deviations as well as to make future improvements. Reviews are divided into two categories, project reviews and product reviews. This report will focus on product reviews because these (the reviews that are presented in this report) reviews can be used as defect prevention techniques, see section 3 for more details. The two review variants that are presented in this report are walkthroughs and software inspections. Pair programming is also presented in this report, because it can be used as a defect prevention technique. In subsection 2.4, a comparison between the presented review variants and other variants are made as well as a comparison between all presented techniques (walkthroughs, software inspections and pair programming) in this report.

Figure 1 shows an overview of how the different techniques are related.

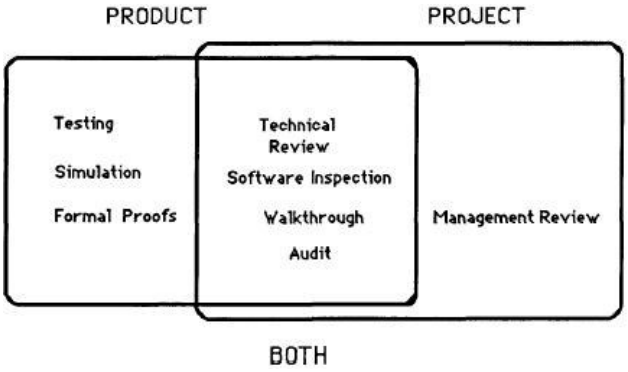

Figure 1 Overview of the techniques [7]

2.1. Software Inspections

An inspection has the purpose of examining a software product to check if the software product conforms to specifications and standards. It also has the purpose to collect data during the inspection for measuring the quality of the software product. Inspections verify that software corresponds to the specification and non-functional requirements. Inspections also verify that the software is inline with regulations, standards and

procedures. Detected defects is also used to make improvements for future inspections by updating strategies and checklists with new defects found that not have been looked for earlier. Inspection can be seen as defect detection and we agree upon that it is defect detection. However, to be able to make defect prevention, one need to find defects to be able to prevent them in the future. During an inspection, ideas for how to solve identified defects are given but no decision is taken of how the defect will be solved. It is more an action of giving suggestions of how to solve the found defects rather than decide upon how the defects will be corrected. The artifacts that can be a target for inspections are requirements specifications, different design documents, source code, test documentation, manuals for users and installation manuals.

When inspections are performed there is a suggestion of being three to six people [1]. Also, these people should have different roles and responsibilities for the inspection.

2.1.1. The roles. The different roles in an inspection are inspection leader, recorder, reader, author and inspector [1]. The responsibility of an inspection leader is to make all the administrative work before the inspection. This can be actions like planning and making preparations. During the inspection he is the one to make sure that the inspection is performed as planned and meet the inspection objectives. He is also responsible for collecting the inspection data and to address the output from the inspection. The output is more described in section 2.1.3. The recorder manages all the recording during the inspection. He documents all deviations that are discovered during the inspection. He also documents all action, decisions and recommendations. The recorder can also be the same person that is the inspection leader. The reader is the one to guide the inspection team through the inspection. He will be prepared on the software product to guide the team in a logical way through the product. He will also know what important areas that is critical to inspect. The author has the duty to make sure that the planning for the inspection meets the software product in a good way. He has to know more about the product to support the planning work as he is the one to gather the inspection material to the inspection leader so ha can plan the inspection. During the inspection he makes sure that all planned work is complete before the inspection is over. The inspector has the task to identify deviations in the software product from different specifications. It is good to have different areas to inspect related to one single inspector. This means that one inspector should concentrate on certain defects while the other inspectors concentrate on something else. In this way you can get a good coverage of all areas.

2.1.2. The inspection. Before the inspection starts there have been some planning work by the inspection leader. First of all he has put together the inspection team and divided the different responsibilities to the team members. He has also spread the inspection material to all team members so they can prepare for the inspection and also to provide feedback on the material before the inspection. Also, the author will present the overview for the procedures for the inspection. Moreover, the inspection leader also schedules the time for the meeting and the place where the inspection will take place.

Except from the planning of the inspection, there are some other preconditions before the inspection can start. The first condition is that the objectives for the inspections have been stated. The second condition is that all the inputs for the inspection are available to be inspected. These inputs are besides the already mentioned objectives for the inspection; the software product, documented procedures for the inspection, forms to report on and existing known deviations. These inputs are the minimum. Other preferable inputs can be checklists, standards, guidelines and specifications. You can also use hardware specifications and performance data for computers that the system will run on. Beside these preconditions for starting with the inspection, there are some other issues to think of before starting with the inspection. It is preferred that the software product that will be inspected is complete or at least the parts that will be inspected are complete. It is also good if there are automated error detection tools available.

As said before, the inspection team members should give feedback to the inspection leader on the inspection material. To be able to give feedback on the inspection material they all examine the software product before the inspection takes place. All deviation from specifications etc is documented and sent back to the inspection leader that will document all findings. This is done to make the time for the inspection as effective as possible. During the inspection the team can focus on more hard found deviations rather than the easy found before.

When the time has come for the inspection to take place there is a suggested agenda to follow [1]. The description of the inspection will follow this agenda. First an introduction meeting is held. Here all the inspectors are introduced to each other and the inspection leader informs about the purpose with the inspection and reminds the team participants to focus on deviation defects rather than on analyzing the defects. Information that the reader documents the finding is told and that the author can be questioned for issues about the software product is stated. The inspection leader also answers questions that the inspectors have that are crucial for the inspection. The inspection leader will also make sure that all inspectors are well prepared for the inspection. Otherwise he should schedule another time for the inspections just to make sure

that the inspection will give the best result as possible. If the inspectors are well prepared, the common deviations are presented for the inspectors to give them hints of what to look for.

When this is done the inspection starts. During the inspection the inspectors examine the software product systematically and objectively. All findings are reported to the recorder that documents the findings on the deviation list created by the inspection leader. If there are any difference in opinion the finding are noted and later discussed in the end of the inspection meeting. At the end of the inspection meeting, all findings are evaluated so that they are accurate and complete documented. Also as said before, in this stage all disagreed deviations are discussed and resolved.

Before the inspection ends, the inspection team decides upon whether or not the software product is approved. It has to be decided on re-work should be done and verify that the re-work will solve the deviations. Before the inspection ends it also has to be a re-inspection schedule to inspect all changed work in the software product. The inspection leader also has to verify that all planned parts in the inspection have been carried out. The inspection is declared complete when all parts of the agenda for the inspections are finished and the outcome of the inspection is presented in a report. The outcome of an inspection is described in the next section.

2.1.3. The outcome of the inspection. The outcome of an inspection process is a document which should cover the following parts [1]; a description of the inspected project, all team members of the inspection and the time they put into the inspection. Also, it should contain a description of the product inspected and the size of the inspected material together with all the inputs for the inspection. Moreover, all objectives for the inspection should be documented in this outcome list together with all found defects described with location, classification and a description of the defect. There should also be a summary of the number of defects found and a dividing into the different defect categories. How the inspection was performed should also be stated as well as the effort spent on preparations. All re-work that will be made should also be stated in this document. Additional outcome of an inspection is data colleted for analysis. This data is used for measuring the quality of the product, measure the effectiveness of development together with maintenance and also effectiveness for the inspection. These measures are collected for the purpose of improvements. All data collected are categorized. A few of these categories are the following; ambiguous, inconsistent, missing, not conforms to standard, not able to implement and risk-prone. Besides dividing all the collected data into these categories, the data is also prioritized into levels. These are major and minor. When the collected data is analyzed, all findings that are frequently occurring as defects should be added to the checklist used during inspections. Also, the data related to inspections should be analyzed to be able to make improvements for future inspections.

2.2. The walkthrough

The walkthrough is often related to code examinations, but a walkthrough can also be used to control interface specifications, detailed designs, change control procedures and test specifications or procedures [2]. Software requirements specification, software user documentation, maintenance manual and system build procedures can also be controlled by a walkthrough [1]. A walkthrough could be used to exam a specific product if the technology is complex or new, if there is a big volume of materials that should be reviewed and a walkthrough could also be used when cost and risk for a product is reviewed [2]. But, if the software is a critical system, walkthrough should not be used as the only review technique.

The purpose with a walkthrough could be to educate personnel about a software product or a new technology as well as to train the participants. It could also be to appraise a software product, to make sure that a product follows the agreed specifications and that the product follows applicable standards. Other important purposes with a walkthrough are to find anomalies and to improve the software product. A walkthrough can be seen as a defect detection technique and we agree upon that it is. However, to be able to do defect prevention, the defects needs to be found to be able to prevent them in the future. A walkthrough could be used to educate and train personnel about a product or a new technology, which can lead to prevented defects in future products. The data that is collected during a walkthrough should be constant analyzed to improve the walkthrough review and to improve the software activity or process that is used to produce a software product. This can also lead to prevented defects for future products that will use the same activity.

When walkthroughs are performed there is a suggestion of being two to seven people [1]. These people should have different roles and responsibilities.

2.2.1. The roles. There are different roles that are needed for walkthroughs. These roles are walkthrough leader, recorder, team member and author [1]. Since a walkthrough can be performed with only two people, one person can have more then one role. For example, the walkthrough leader can also be the recorder or the author.

The walkthrough leader has the responsibilities handle administrative tasks, like distribute documents and plan the meeting. The walkthrough leader has the

responsibility to make sure that the walkthrough is performed in a right way and to prepare the objective statements for the team. The walkthrough leader also has the responsibility to make sure that the right outcomes are produced. These outcomes are described in more detail in section 2.2.3.

The recorder has the responsibility to document all decisions and actions that is discussed during the walkthrough meeting. The recorder should also document comments about found anomalies, contradictions, improvements for the activity or other approaches for solutions during the walkthrough.

The author's responsibility is to present the software product during the walkthrough.

Team members have the responsibilities to review input materials to the walkthrough, participate during the walkthrough and to make sure that the walkthrough meets the walkthroughs objectives.

2.2.2. The walkthrough procedure. Before the walkthrough can start, two preconditions must be fulfilled. The first condition is that the statements of the objectives should be defined and the second condition is that all required inputs like the software product and statements of objectives are available. There are four phases in the walkthrough procedure [2]. These phases are planning, overview, preparation and examination.
It is the walkthrough leader that is responsible for planning the walkthrough. To do this, the walkthrough leader should identify a team, schedule and decide upon a place for the meeting and to distribute needed input materials to all team members. In the overview phase, the author should present an overview picture of the product during the walkthrough meeting. In the preparation phase, all input materials should be reviewed. All found anomalies during the review should be sent to the walkthrough leader before the meeting to make the meeting more effective. In the examination phase, the meeting starts with an introduction to this meeting, the purpose of the meeting, that the members should focus on detection and not comment the author, only the software product. After the introduction, the author gives the overview presentation and then walks through the software products elements. During this walkthrough, team members ask questions and rises issues about each element of the product as well as taking notes. After the walkthrough, the leader leads the discussion to decide upon alternative solutions or actions. The recorder takes not from the discussion. The walkthrough team should also decide if there is a need for a follow-up walkthrough. The meeting is completed when all product elements has been discussed. After the meeting, the walkthrough leader should issue the report with the right output; see section 2.2.3 for more details.

2.2.3. The outcome of the walkthrough. The outcome from a walkthrough is documented evidence that should include the following parts [1]; all team members from the walkthrough and the examined software product. Also, it should include the objectives statements that were supposed to be achieved during the walkthrough together with information if they were achieved or not. There should also be two lists, one that contains recommendations for each anomaly and a second that contains actions. Each action should have information about due dates and a responsible person. There should also be information about any needs for a follow-up walkthrough.

2.3. Pair programming

Pair programming has been nominated and acknowledged many times in the past as an improved way of developing quality software [9]. Pair programming is a practice or a custom in which two programmers sit side by side on a single work station or a computer. This team of two programmers continuously collaborates and works together on the same design, code, algorithm and test [8]. One of the team members/programmer can be referred as the 'driver'. This has the control of the keyboard/mouse and is involved in the active implementation of the program and the code. The other programmer known as the 'observer' is involved in continuous active observation of the work of the driver. The observer looks for the syntactical and spelling mistakes and is also involved in the direction of the part of project for future. He is also involved in looking up the resources, considers strategic implications of the work and asks questions to be answered. Both the observer and the driver can and mostly switch roles and work together and have equal share in the development of the program they work on. They also are involved in effective communication and can work on challenging problems and can brain storm them.

2.4. Comparison of techniques

In this section we will compare different techniques. First, there will be a comparison between the techniques described in this report. We will compare software inspections, walkthroughs and pair programming and describe the differences between them. Moreover we will make some comparison between the different variants of review techniques. We will describe the differences between our described techniques, software inspections and walkthroughs against the other variants like management reviews and technical reviews. Here we do not focus on pair programming as we do not see that technique as a review or inspection. However, when doing the comparison we think of walkthrough and inspections

as one single unit in comparison to the other variants of techniques.

2.4.1. Between the described techniques. The walkthrough and the software inspection review techniques have a lot in common. For example, both are in the product review group, both techniques collects data during the walkthrough and the inspection and that no management people should participate during the walkthrough or the inspection. Other similarities are that both techniques are defect detection techniques and both can be used for defect prevention in future projects and future development of products.

Even though there are many similarities, there exist some differences between the walkthrough and the software inspection [1] [2]. A software inspection performs a more thoroughly evaluation of some parts of a software product, while a walkthrough evaluate the whole product. One of the walkthrough objectives is to consider alternatives for a specific solution or functionality and so forth. In software inspections, alternatives should be ignored. A walkthrough can be used to examine new or complex technologies because there is no need for expertise people in a walkthrough. One purpose to use a walkthrough for new and complex technologies is that the personnel can be educated after the walkthrough. This is one of the objectives in the walkthrough. A software inspection is not suitable for this task because a software inspection needs a team with right expertise knowledge as well as trained staff. Another difference is that software inspections uses checklist during the inspection, while there is no use of checklists in a walkthrough. If data needs to be collected, even though both techniques collect data is software inspections to prefer, because a software inspection can have more expertise personnel with expert knowledge about the product and performs a more thoroughly evaluation.

Pair programming is not like walkthroughs or software inspections, because pair programming is a defect prevention method while walkthroughs and software inspections are more defect detection. But the result from these two methods can be used to prevent defects to enter the system later in the project and in future projects. However, both software inspections and pair programming works on a single problem thoroughly and comprehensively evaluates the part of the problem.

However unlike software inspections the alternatives are not ignored and all possible solutions to problems are well thought, discussed and analyzed and then the best possible choice is implemented and used. Another difference between software inspections and pair programming is that the members of the pair need not to be highly skilled people. The teams could be a mix of one experienced person and one junior or fresh person.

2.4.2. Between different variants of reviews. The different variants of the techniques that we compare software inspection and walkthroughs are management reviews and technical reviews. These two last mentioned techniques are very similar to each other but what separates them are that management review aims for evaluate a project to clarify its status and progress [1]. The aim of technical reviews is more for the purpose of evaluate the products status and conformance to specifications [1].

To start with comparison between our mentioned techniques against management review, we can say that management reviews cover more material to examine. The purpose with management reviews as said are to evaluate the project while our techniques have the purpose of find defects and deviations from specifications. Also, our techniques can be used for preventing actions for bringing faults into the product while management reviews just examine the current status of a project. Our techniques can be used for defect prevention but management review just detects problems. During a management review, management people are involved in the work in comparison to our techniques where more technical people perform the reviews. Also, management reviews uses more management documentation for evaluation while our techniques study more technical documents.

To compare our techniques against technical reviews we can say that also in this comparison, more material are examined in a technical review. Technical reviews does not concern ay data collection or improvement work as our techniques but just aims to examine the status of the product and the conformance to specifications. Technical reviews also just detect faults. We think that another difference can be that our techniques can be used earlier in a project to check requirements specifications and design documents while technical and management reviews are performed a little later in the project.

3. Prevented defects with this technique

In this section we will describe different defects that can be discovered with these techniques and also how these found defects can be prevented. We will also present metrics for how these preventions can be measured in terms of beneficial for the prevention. Last in this section we will also present three defects from past projects that we have been part of and how they where prevented both from coming into the code and how they were prevented in future projects.

3.1. Walkthrough

When performing walkthroughs, there are different defects that can be found from source code,

documentations and some procedures. But, we think that walkthroughs are most suitable for requirements and design walkthroughs because there is no or almost none preparation. To walk through requirements and to walk through the design description, not much preparation is needed. But, to check the source code a good understanding and knowledge about the product is needed. Of course can the source code be walked through, but we think that software inspections are better to check the source code because it performs a deeper inspection, see section 3.2.

In the requirements specification, a walkthrough should look for defects like omissions, unwanted additions and contradictions [2] between requirements. But, this is not all; a walkthrough should also suggest and consider other suitable functionalities and performance objectives [2]. This is done to see if there are better solutions that will prevent defects from coming in to the sharp requirements specification and the implementation.

In the design description, a walkthrough should look for defects like problems with the current design, missing functionality and problems or bad design solutions. But, even here a walkthrough should consider and suggest other suitable design solutions.

Besides from these defects, a walkthrough can be used to check a product technology that is new or complex. The reason for using walkthroughs is that it could be hard to find a good review team with the right expertise [2]. The purpose to have a walkthrough of new technologies or complex technologies could be to educate the rest of the team members after the walkthrough. If the team members get more knowledge about the technology, defects can be prevented to come into the implementation.

Walkthroughs can use the same metrics as software inspections and collect the same metrics as well, see section 3.2 for more detail information. Walkthroughs can use and collect metrics, but it is recommended to use software inspections when metrics should be collected [2] since software inspections are more thorough.

3.2. Software Inspections

When performing inspections, the different defects that you can find are defects from documents and source code. Except from spelling errors and deviations of standards, there are many other defects to look for. Defects from the requirements specification are most likely to be defects like, conflicts between requirements and bad stated requirements. As there are no other documents that can verify requirements specification, it is hard to find that many variants of defects here.

Defects from design documents that can be found are missing functionality that can be traced from the requirements specification. You can also look for design solutions that might be problematic once implemented. It is good to check for nice dividing of the system with interfaces between the parts. Inspection of design documents should focus on finding solutions that might be casual so that design keeps as simple as possible.

Test documentation is another document well worth to inspect. Here you can examine that all functionality is tested in the system. It is the matter of making sure that all requirements are covered in the testing phase.

During code inspection there are a lot of defects to look for. First of all, checking that the code standard is followed is quite easy but maybe trivial. Other defects that can be found during code inspection can be divided into categories. These categories are described further. The first category is data faults and here do all faults like instantiation of variables before use belong. Also, making sure all variables are used. Then we have control faults. These are faults that derive from loop non-termination, not all cases are considered in statements. Input and output faults are another category of faults. These are faults originated from unexpected inputs to a function. Also that outputs from a function might not have been assigned. Then we also have interface faults. Parameters for a function might have been stated in wrong order or there is not correct number of parameter for a function. Storage faults are another category where memory allocation is considered and last we have exception faults which mean that all error exceptions might not have been taken care of.

Faults found during code inspection might not be seen as defect prevention. In the current project it is not but when these faults are collected you can analyse which the most common errors are brought into the code and then try to found their origin, i.e. why they were brought into the code. If you can discover this source then you can be used to change procedures to prevent the faults from appearing that often in the code in the future.

Software inspections can be used to collect metrics from documents (requirements specifications, quality manuals, design documents, test documents and so forth) and source code. Metrics that can be collected in documents are spelling mistakes, grammar mistakes, and conflicts between requirements, procedures, design solutions and so forth. Moreover, can duplications, ambiguous requirements, test cases and so forth and contradictions in the documents can be collected. In the source code can the following metrics be collected, deviations from code standard, minor coding errors divided into different categories and bad structured code that are hard to read/understand.

3.3. Pair programming

As described earlier pair programming is more or less like software inspections. Pair programming involves

continuous reviews of design and code that leads to most effective defect removal rates. Also with pair programming deviation from standards and the syntax and semantic errors are more likely to be found and removed. This saves a lot of time on part of recompiling the code and looking for missed commas, function missing braces and non-initialized variables etc. These sorts of defects are more likely to be found using pair programming because it involves one person continuously looking over the monitor for the code that is being written. Also previously described the different categories of defects that can be checked using software inspections can not only be detected but also prevented using the pair programming. Therefore pair programming can server as a defect prevention technique in those cases making those defects not appearing in the code in the first place.

Also with pair programming defects in the design can be thoroughly reviewed, inspected and analyzed. With pair programming a simpler and a superior hybrid design is produced because of the arguing of the team members within the pair.

In addition to that with pair programming test cases are produced initially and refactoring is done while the code is being written. This ensures that the design would be simple and the code would be manageable and precise thereby leading to fewer errors in the design and test phases.

3.4. Three defects from a past project

The first defect comes from using the JSP/Servlet programming language. The problems were that it took a very long time to search for defects in the source code, it took time to redo the work and to test the new code. Other problems were that after the delivery there was still errors in the interface like wrong language, spelling mistakes and graphical faults. In a large project where the same two authors of this report participated in, JSP/Servlet was one of the programming languages. The difference was that in the large project was pair programming used. This prevented many defects to enter the system. There were fewer compiler errors (the feedback was bad even here), which lead to less time spent on searching in the source code for errors, reworking and testing the system again. Another important aspect was that there was no spelling mistakes, wrong language or any problem with the graphical part after the delivery. This saved a lot of time because no time to rework or producing a new deliverable product was needed.

The second defect is an example where a technique was used to prevent defects to enter the system. The defect was a design error and was removed by performing a walkthrough. In the design of the software product there was a big error that would have removed the possibility to run the simulator with multiple simulation engines that should act like a central for sending messages via mobile phones. The project group spend one day to go through (like a walkthrough) the design of the system and found this design error. This prevented a lot of defects to enter the system. Since it took three to four weeks to build the right simulator, we think that this preventive action saved at least three to four weeks.

The third and last defect was not prevented, but it could have been prevented and saved a lot of time for the project members. When the simulator was completed (one part of the whole project) and should be tested against the customers simulator, there was a lot of errors that occurred. The messages signals and data from our simulator could not be interpreted by their simulator and our simulator could not interpret their signals and data. The customer thought that we had a problem in our simulator. After two weeks of testing and a lot of reworking where many defects came into the system, it turned out that the problem was on their simulator. Their simulator did not follow the different standards from the operators of sending messages from mobile phones. If the customer had at least spent one day to inspect their simulator to see if it follows the standard, we could have saved two weeks of testing and the time it took to restore the simulator.

4. Acceptability of the techniques

In this section we will discuss how the techniques can be motivated to perform. We will present different motivations for different kinds of stakeholders in the organization. There is a need to use different kinds of motivation for upper management than it is for developers. Also, in this section, resistance to the techniques is described and how the resistance can be met and managed.

4.1. Software Inspections and Walkthrough

When performing software inspections and walkthroughs there might be some resistance of the work as developers might see the work as you check so that they have not made any mistakes. Also to make it possible to perform reviews it requires that management and developers are made aware of the benefits they can earn from the reviews. Motivation and resistance of reviews are more described in the next two sections.

4.1.1. Motivations. As a major motivation for performing inspections and walkthroughs, defect detection and defect prevention can be used. Defects detection early in the development process prevents faults from going into the

source code. The more faults that can be prevented from going into the source code the less faults will end up in the product that goes to the customer. We think that testers will find about the same amount of faults when testing the system. Of course, performing inspections and walkthroughs are expensive as you dedicate a group of people that does not develop something for about a day. Expenses should be kept for oneself [4]. However, what this group can detect early saves time and money in the end as in the future, processes might have been improved to prevent common found faults or developers have learnt how to prevent these faults. Inspections and walkthroughs makes people in an organization think more in terms of quality, especially if they can see the benefits from performing these quality checks.

When performing the inspections and walkthroughs, it is important to collect data that can be analyzed and serve as feedback to the people in the organization. When they see the benefits they will be more cooperative [4]. Inspections can also be used for the purpose of verify and validate documentation. This is an important aspect to actually verify that you develop the correct system and also to check so that the system is correct implemented. Also, when performing these reviews of a software product, people in the evaluation team get more knowledge of how the product works and are built. New people brought into an organization or people new to a project or product can be a part of the reviews to learn about the product.

4.1.2. Resistance. Resistance in an organization against reviews, we think is very common. Developers do not like that someone else checks that they have done a good job. They might think that reviews are to be done for the purpose of finding those people that are less good in programming and bring in more faults to the system. Also, they can feel that it steals time from them if they are forced to participate in the work. Also, not all people are fond of doing this kind of work and really think it is boring. They may also know that these reviews can lead to improved processes in the future which may force changes of their way of working. People often do not like changes.

How to overcome this resistance is to make the developers understand that these reviews are not for making corrective actions for their bad work. It serves the purpose of measure the quality in the product and in the process together with earning knowledge of how to prevent defects in the future. It is important to provide feedback both to developers and to management to prove that all reviews are beneficial, that the whole way of working has been improved and that the quality of the product has been improved. This should make it more enjoyable to work in an organization that develops quality products.

4.2. Pair programming

Pair programming is now being widely used in the industry as well as academia and many experiments have been performed to check its effectiveness. Like every other technique it faces resistance on part of the people involved in using it and has been discussed in the next two sub-sections.

4.2.1. Motivations. Like software inspections and reviews, the major motivation behind pair programming is defect detection and prevention. With pair programming fewer defects are introduced in the design and code. In addition to that people working in pairs find the experience more enjoyable than working alone. This also enhances the satisfaction of the programmers as they produce better solutions and work more effectively and efficiently. Also with pair programming there is effective communication involved within the pair members [10]. This helps in producing better and faster solutions to difficult problems and improves the over quality of the system. Also with pair programming there is learning going on at all stages as both the persons teach each other and learn from each other thereby increasing their problem solving skills. These are some of the factors that can be used to motivate the developers, managers and organizations to use pair programming.

4.2.2. Resistance. As people do not like change, there is also a resistance of using pair programming within the organizations. One of the major resistance factors within the organizations and the managers is the economic factor. The affordability of pair programming is one of the key issues which the managers need to deal with. If it would be too costly then the managers would feel reluctant to use it. There is an assumption that the introduction of pair programming increases the development costs and manpower overheads. The author agrees with the assumption but one thing should be kept into consideration that pair programming reduces the number of defects that are being introduced in the beginning. This helps in minimizing the cost of redoing the work and detecting problems at the later stages of the development and testing. The empirical evidence shows that removing and fixing defects at later stages are much more expensive [11], [12].

5. Strengthens and weaknesses

In this we will present some different strengths and weaknesses for software inspections, walkthrough and pair programming. There are many advantages and disadvantages for all the techniques and should be

carefully analyzed when the different technique are most appropriate to use.

5.1. Walkthrough

There are both advantages and disadvantages with walkthroughs. We will start to present some advantages with the walkthrough. The major advantage with the walkthrough is that it can be used to go through a lot of material rather fast [3]. This means that a whole product or a requirements specification can be reviewed rather fast. By reviewing the requirements specifications and to suggest alternative functionality if something does not work, walkthroughs can prevent defects to come in to the implementation.

Another advantage with walkthrough is that there is almost no need for preparation [3]. There are some participants that need to prepare a little, but not everybody and not like in software inspections (see section 5.2 for more details). Since there is almost none preparation, walkthroughs are suitable to use when the technology is new or complex [2]. The new technology can be walked through and the participants can then educate the rest of the team members. This is also a way to prevent defects to come into the implementation phase. Another advantage is that a lot of people can understand the information since it is not that deep and detailed [3].

One disadvantage with the walkthrough is that the people working or at least are tightly coupled with the product could be bored during the walkthrough. Completely new people to the product may not catch up with the rest of the people. In both cases, chances of stating interesting problems and suggestions can be missed out. Since no advanced preparation is needed, the participants have different understandings and different knowledge about the product [3]. This can lead to discussion where the participants do not understand each other and problems might not be found. The participants with the best knowledge and understanding of the product are probably people that are working with the product or close to the product. These people might be blinded by the defects in the product since they see it every day and can not come up with better suggestions. If then the rest of the participants do not have a good understanding and knowledge about the product, they might not be able to see the problems or the bad solutions.

Another disadvantage with walkthroughs is the collected data. Since the walkthrough do not look deep into the product, the data might not be complete and therefore should not walkthroughs be used when metrics should be collected [2]. Software inspections are more suitable for this task.

5.2. Software Inspections

Inspections have both advantages and disadvantages as anything else. We will start with discuss the advantages before we move on to the disadvantages. The major advantage with inspections is that it finds defects on the deep. Especially with known software products that is not so complex. For new technical software product that is complex it is more suitable for doing reviews [2]. This is more discussed in section 2.4.1. As we see it inspections are very good for checking the source code. Many faults can be found here. These faults can be collected for the purpose of educate so that they are prevented for going into the system in the future. Inspections of documents provide a good prevention of bringing faults into the implementation as many faults are removed in an early stage of development.

Inspections of documents that prevent errors from going into the implementation of the software product give advantages like more efficient testing as many faults are already removed earlier which makes testing more focused on the remaining faults in the system. If fewer faults are in the system when testing, the test team deals with those faults and less faults are in the product when delivered to the customer. When performing inspections, the software product does not need to be implemented and the found faults are prevented from getting into the code. Another advantage is that inspections find the fault and the location of the fault in comparison with testing that only finds the fault. This makes an inspection more efficient as time can be saved. When inspections are focused on both documents and source code, there is a possibility to faults that have been covered by other faults when testing the software product. When testing, one error can be discovered which are caused by two faults. It is a risk that just one of the faults is found when doing correction which leads to more time spent on testing. When doing inspections you can find these two faults before they go into the system. Also, when inspections are performed it uses old known defects as input so the team has some clue of what to look for. Moreover, a checklist is used during the inspection with previous known defects which makes it easier perform the inspection as you can follow the checklist. This checklist is updated for each inspection which makes common defects visible in this checklist. Also, the team that performs the inspection has made previous inspections and has some clue of which defects to look for and where to look for them. They are aware of earlier mistakes. All inspectors can also divide certain defects among them so that all previously common defects. This makes the inspectors not look for the same defects which makes the inspection more effective and also more complete.

The disadvantages with inspections are that the checklist might be used too intensively and inspection might just focus on the most common defects rather than also find other defects. Developers also might feel that they are criticized by having inspections for checking their work. Also they can feel that inspections are looking for the one that brought in the defect into the document or source code. This can lead to resistance against the inspection work and can also lead to that no one likes performing inspections. Inspections can be seen as not so exciting work from the reason of resistance to the work and this can lead to non-effective inspections. Inspections can be hard to motivate as they are expensive to perform in the beginning before they are proven beneficial. Moreover, inspections remove defects that never will be visible which makes it hard to prove that inspections are important. Inspections need much preparation before it can be performed. Organizational standards need to be learned and code standards. This makes an inspection more expensive because of the preparations. Also, when performing the inspection it is hard to check that the software product conforms to the customers and users need. It is only possible to check against requirements specifications. Then there is also a problem to check non-functional requirements. Non-functional requirements we think need to be tested to actually prove that they meet the requirements.

5.3. Pair programming

Like every other technique pair programming has its advantages and disadvantages. In this section first the advantages has been discussed followed by the disadvantages.

One of the biggest advantages of pair programming as mentioned earlier is the introduction of fewer bugs in the initial stages of development. This helps in minimizing the redo cost. It also helps the testers to stay more focused towards bigger and major problems that need to be tested within the system.

Another advantage of pair programming is collaborative work and ego-less programming along with comprehensive communication. Every one owns the work equally and works equally to produce better, effective and efficient solutions to the difficult of problems.

Another major advantage of the pair programming is the communication over-head between the team members. Two people can easily communicate as they are closely coupled with each other within a single boundary and space. This helps in producing better solutions and decreasing defects and helps in defect removal and prevention.

The disadvantage of pair programming is that people might not like to work in teams of two. Also the effectiveness of pair programming is dependent on the effectiveness of team members. If one of the team members is not reliable pair programming can produce worst results than individual programmer. It also introduces resistance on part of the programmers and the management as it is considered to be an overhead for the organization to deploy two people to carry out a single task.

Another disadvantage of pair programming is that there is not enough conclusive empirical evidence of its effectiveness in the industry although some evidence in academia is present [13].

6. Relations to ISO and CMMI

This section discusses the different techniques and the relation to the quality thinking standard TickIT and the model CMMI. We choose TickIT from the ISO standard as TickIT is more directed towards software engineering. We will present what TickIT and CMMI advice about the different techniques, software inspection, walkthrough and pair programming.

6.1. Software Inspection and walkthrough

The capability maturity model integrated [5] has some goal that are inline with the described review techniques. On the second level of capability maturity model integrated, it is recommended that activities for controlling and monitor projects and products are carried out. Also, on the same level, you should perform reviews on process and product. On the third level it is suggested that verification on the work is performed. These suggestions are inline with our techniques in terms of defect detection. On level three, you should also collect measures for improvement purposes but, it is not before entering the fourth and fifth level of the capability maturity model integrated, that you run into defect prevention and improvement on processes. As we see it, improvements and defect prevention is nothing you do in the early stage of the capability maturity model integrated. For the lower levels it is just detection that is the aim for reviews. However, detection is good as it removes faults early in the process. Still, just because that capability maturity model integrated mention prevention on the upper levels does not necessarily mean that it is forbidden to do it even on lower levels. Prevention can always be made even if it is just small actions. It does not have to concerns large changes in an organization but may concern changes of tools or some other small actions.

The TickIT standard has some parts that are inline with the described techniques in this report. TickIT standard has design and development review as one of their demands. This means that the design and development

should be evaluated and that problems should be identified together with suggested actions. The results should also be recorded. This demand is inline with our technique in terms of defect detection and as well as the list of suggested actions in walkthrough. Both walkthroughs and software inspections keep records of identified defects and actions. Another demand from TickIT is to have corrective actions. An example of a source that should have corrective actions (which means to eliminate the defects actual cause) is product defects. This demand is inline with our techniques in terms of defect detection.

7. Conclusion

To conclude this report we say that inspections and walkthroughs are for defect detection. However, to be able to prevent defects in the future you have to detect defects in the past to actually know what to prevent. It is the most common defects found by doing inspections and walkthroughs that is the base for future preventions. Moreover, as inspections in early phases of development finds errors that are prevented from coming into the code this can be seen as defect prevention.

Walkthroughs are good to use when systems are complex and when the technology working with is new. Also, walkthroughs are good for go through large amount of material while inspections are more thorough and there are a need of dividing inspections for inspect smaller parts of a system or development. Also, inspections are better for collecting data as it has a more thorough focus.

These techniques are not that strait forward. Resistance to these techniques within an organization is not that uncommon. Developers may think that the purpose of these techniques is to check their work. However, it is important to give feedback on the purpose and also the result from the inspection.

The strengths with these techniques are that walkthrough needs less preparation to perform. Furthermore, no one needs to be an expert in the area. The strengths with inspections are that the source of a fault is found instead of finding the error while testing and then have to search for the source. The weaknesses are for walkthroughs that, different people can have different levels of knowledge which might lead to that certain people are bored during a walkthrough. People might have difficulties to understand while some think that certain things are trivial. For inspections there is a problem of people that do not see the meaning as faults are removed before they turn up as an error in the code.

8. References

[1] IEEE Standard for Software Reviews, retrieved on the 6 December from http://ieeexplore.ieee.org.miman.bib.bth.se/iel4/5362/14498/00666254.pdf

[2] Hollocker, C.P, Software Reviews and Audits Handbook, John Wiley & Sons, United States of America, 1990.

[3] Freedman, D.P, Weinberg, G.M, Handbook of Walkthroughs, Inspections and Technical Reviews: Evaluating programs, projects and products, Dorset House Publishing, New York, 1990.

[4] Gilb, T, Graham, D, Software Inspection, Addison-Wesley, UK, 1993.

[5] CMMI for software engineering, retrieved on the 6 December from http://www.sei.cmu.edu/publications/documents/02.reports/02tr028.html

[6] Executive overview of TickIT, retrieved on the 6 December 2004 from http://www.tickit.org/overview.pdf.

[7] IEEE standard for software reviews and audits, retrieved on the 6 December from http://ieeexplore.ieee.org/iel1/2366/1231/00029123.pdf

[8] Pair-programming, retrieved on 15 December from http://www.pairprogramming.com

[9] Beck K., Extreme Programming Explained: embrace Change, Addison-Wesley, Reading, Massachusetts, 2000.

[10] L. A. Williams and R. R. Kessler, “All I Ever Needed to Know About Pair Programming I Learned in Kindergarten,” in Communications of the ACM, vol. 43, no. 5, 2000.

[11] Capers J., Software Quality, Addison-Wesley, 1996.

[12] Humphrey W.S., Introduction to the Personal Software Process, Addison-Wesley, Reading, Massachusetts, 1997.

[13] In Support of Student Pair-Programming, retrieved 15 December from http://www.pairprogramming.com/WilliamsUpchurch.pdf