Source Code Inspection

Measuring the Difference Between Individual
and Group Performance

Andreas Dybdahl

Master of Science in Computer Science
Submission date: June 2013
Supervisor: Tor Stålhane, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Source Code Inspection – Measuring the Difference between Individual and Group Performance

Abstract

The purpose of this study is to examine the relationship between the performances of group efforts versus individual efforts. Study of previous research on software inspection has shown that opinions are divided on whether to have a group work stage in the inspection or not. Initially believed to be beneficial for finding defects in source code it has in more recent years shown to be not as effective as once believed and possibly not worth the cost.

This study examines the result of an experiment that conducts a slightly altered version of software inspection where subjects work either solely in groups or solely as individuals where the traditional method has subjects first inspect the code alone and then forming a group to discuss their findings.

The hypothesis was that the subjects working in groups in such a fashion will perform better, but it is shown that neither group of subjects outperforms the other at finding defects. These results are consistent with previous research in the field, however it is interesting to note that even without individual preparation groups perform similarly to how they have performed in other studies. Since one of the chief arguments against group work is its high cost eliminating a stage from the review process would make it less costly and thus perhaps more desirable as an inspection method.

Acknowledgements

I would like to thank my professor Tor Stålhane for his tremendous patience and assistance in helping me complete this work. His knowledge and experience with statistical analysis was of great help to me during this time.

I would also like to thank my family for always being there for me.

Contents

List of Figures

List of Tables

Chapter 1 – Introduction

1.1 – Software Inspection

The subject of this thesis is an area of expertise called software inspection. In the world of software development software inspection involves itself with developing software with as few defects as possible in a cost efficient manner while still ending up with a product that produces the correct output and conforms to its specifications and the expectations of the customer.

Over the years there have been several developments on software inspection inspired primarily by the work of (Fagan, Design and Code Inspections to Reduce Errors in Program Development, 1976) in 1976. Michael E. Fagan is considered to be one of the pioneers of software inspection. His initial inspection was designed due to his frustration with the software inspection paradigm at the time called a walkthrough. The walkthrough process is fairly simple. The author of a piece of code sits down with another programmer and explains to this programmer what his code does and what it is supposed to output. The second programmer then reviews the code for any defects the original author might have missed or did not think of which the author then fixes after the walkthrough is completed.

The problem with the walkthrough process was that it lacked rigor and had no clearly defined rules. The formality and discipline exerted in a walkthrough inspection varied greatly and ultimately led to the birth of the inspection process by Fagan. Ten years later Fagan reported creating a final product with 38% less defects and a 9% reduction in average project cost (Fagan, Advances in Software Inspection, 1986) for IBM using his inspection over walkthroughs.

After the initial report of Fagan several researchers (Parnas & Weiss, 1985), (Bisant & Lyle, 1989), (Martin & Tsai, 1990) have reported similar findings on the positive effect of software inspection in reducing the amount of defects in a finished product while remaining cost effective due to the fact that less time and resources is required to fix post-development defects.

1.2 – Motivation and Context

This section will focus on the following:

- What I want to find out.
- How I want to find it out.
- Why I want to find it out.

1.2.1 – What

What I want to find out is whether or not the inspection process can be improved by removing the individual inspection phase rather than the group phase of a software inspection. In an experiment by (Stålhane & Husain Awan, 2005) they find that nominal groups generally outperform real groups, however this is not always the case. Similarly (Porter, Votta Jr., & Basili, 1995) discover a 35% increase in defect detection in their experiment using a scenario based approach to defect detection, however when their experiment was replicated by (Lanubile & Visaggio, 1996) they found no such improvement using the methods of Porter et al.

So why does the group phase of software inspection sometimes work and sometimes not. I intend to look closer at whether or not the individual phase of the inspection process might get in the way of the group phase due to the fact that the individual subjects might be too keen to show off the results of their individual preparation instead of focusing on the group work aspect of the inspection which might thus lead to the group phase becoming unproductive for further defect detection.

1.2.2 – How

This brings us to the how to get this done. As a student resources are limited so I intend to carry out an experiment using 3rd year computer science students. Then these students can be divided into two groups: those working as individuals and those working in a group setting.

A short piece of Java code will be handed out to both groups along with an explanation of how the code works and what it is supposed to do as well as an anonymous questionnaire to be filled out upon completion of the experiment to record a bit of data about the subjects to provide further background for the data obtained by the experiment.

After the experiment is completed and the data collected nominal groups can then be constructed from the individual subjects which creates a basis of comparison for the real groups performance.

The group size is set to three people working together or the union of defects detected by three individuals for the nominal groups. This is due to the fact that while it is reported that larger groups generally perform better (Stålhane & Husain Awan, 2005), (Martin & Tsai, 1990) it is also reported that larger groups waste as much as 20% of their time simply getting together (Votta Jr., 1993) and that a group should be as small as possible while still being able to perform its function (Campion, Medsker, & Higgs, 1993). In addition a smaller group size will be less costly to implement in a real life setting and might thus be more desirable to implement and try out by the management as the cost of failure is less should it not work out.

1.2.3 – Why

Rather than rid the inspection process of the group phase due to its flaky performance in recorded research I want to attempt to create a method that results consistently in the good results reported by previous research due to the fact that group work can result in secondary benefits to the company implementing the method i.e., being educational for new programmers as they will get a more solid grasp on the source code by reviewing it with more experienced peers or improving social conditions in the workplace.

1.2.4 – Research Questions

The answers we are in search of in this thesis could more formally be posed as a series of research questions (RQs):

- **RQ1:** Are real groups better than nominal groups when it comes to identifying defects using code reviews?
- **RQ2:** Are real groups better than individuals when it comes to identifying defects using code reviews?
- **RQ3:** If one type of review is better than the others, is this uniformly true or will the conclusion differ depending on defect types?

RQs 1 and 2 will be checked using a t-test and RQ 3 will be checked using a two-sample proportions test.

Chapter 2 – Software Inspection

The origins of a formal software inspection method can be traced back to a paper released by (Fagan, Design and Code Inspections to Reduce Errors in Program Development, 1976) which outlined a disciplined structure on how to conduct a software inspection with the intent of maximizing the discovery of defects.

The process he outlined had the six following steps:

- **Planning:** The inspection team is formed. Fagan suggests four as a number for this team. Each of the members are assigned a role, described later, in which they function.
- **Overview:** The inspection team meets with the designers to learn about how the software is intended to function in detail. Documentation of the work done so far is provided at the end of the overview.
- **Preparation:** The inspection team members sit down with the design and look over it individually doing what they can to understand how it functions.
- **Inspection:** The whole team now comes together in an attempt to find defects in the design. The reader of the team describes how he would implement the design and the rest of the team attempts to find any flaws in his description.
- **Rework:** The fixing of any defects discovered to be done by the author. The process of inspecting and reworking may be repeated as necessary.
- **Follow-up:** The moderator of the team must follow-up the rework to check that if all the defects discovered by the team have actually been resolved.

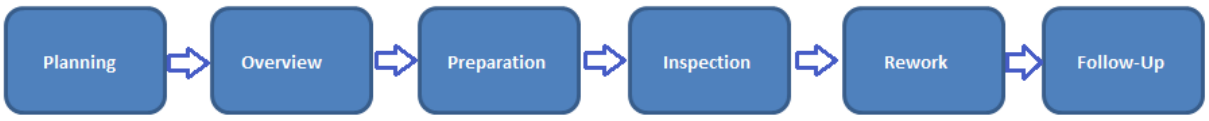


Figure 1 - The Fagan inspection process

In addition to these six steps Fagan also introduced roles in his suggested group. These roles are:

- **Moderator:** The team leader is the most important role for this type of inspection. It is recommended that this person should be a competent programmer, but he does not have to be an expert on the program being inspected. His duties involve the coordination of the inspection process.
- **Designer:** The programmer who produced the design.
- **Reader:** The programmer takes the design and translates it into code for the team to review.
- **Tester:** The programmer who creates test cases for the suggested code/design.

Fagan's philosophy behind suggesting this method of inspection is simply that catching errors early in the design is far cheaper than finding them later on and that a certain degree of formality and discipline is required for these inspections to work. In essence this inspection is a more formalized version of a walkthrough which is simply the programmer sitting down with other members of the development team and walking them through the program and what it does and the other members then ask questions to the programmer in an attempt to find errors and other problems.

Fagan reports 38% less defects in a final product compared to the walkthrough approach and a 9% reduction in average project costs for IBM when using inspections over walkthroughs (Fagan, 1986).

The Fagan inspection served as inspiration for other methods later on, a few of which I will detail here. The methods I will write about are called:

- Active design review.
- Two-person inspection.
- N-fold inspection.
- Phased inspection.
- Meeting-less inspections.

2.1 – Previous Research

This section describes alternate approaches that have been tried after Fagan suggested his inspection. Most incorporate the Fagan inspection in some way, but in an altered form.

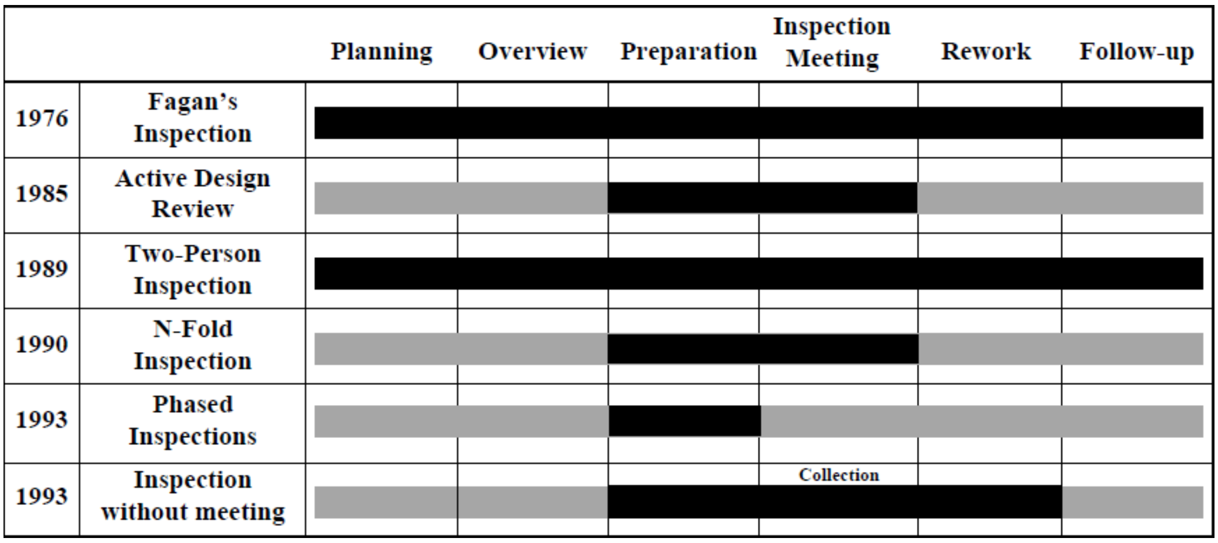


Figure 2 - A comparison of different inspection methods (Aurum, Petersson, & Wohlin, 2002). Black and grey areas denote the parts of the inspection process that have been emphasized.

2.1.1 – Active Design Review

This method was suggested by (Parnas & Weiss, 1985) because they felt Fagan's inspection was not good enough at detecting error in the software creation process because the steps were too overwhelming. The reviewers are simply not that well acquainted with the program they are inspecting and the amount of information they must process in the preparation phase is too much. They thus redesigned the review process.

Instead of having meetings based on trying to understand the process of the entire program, reviewers are instead required to sit down in several, smaller meetings each focusing on a smaller part of the program.

These smaller reviews are organized in three stages. First, they are presented with a short summary of how the program is supposed to function. Second, the reviewers sit down with the material and study it. Lastly, the defects discovered by the reviewers are discussed with the designer.

Reportedly "*Parnas and Weiss successfully applied this approach to the design of a military flight navigation system, but did not provide any quantitative measurements.*" (Aurum, Petersson, & Wohlin, 2002).

2.1.2 – Two-Person Inspection

The two-person inspection was suggested by (Bisant & Lyle, 1989) as a way of easing into a more thorough type of inspection like a Fagan inspection prompted by a conducted survey which showed that 84% of companies surveyed used software inspection, however none of them performed the inspections correctly. They hypothesize the reason for this being a reluctance to try out new measures when they are costly to implement with an uncertain return on investment.

In their experiment (Bisant & Lyle, 1989) report that a two-person inspection method appears to improve productivity, in particular for novice programmers. The experimenters note that the subjects used are 3rd year students and as such the observed results may not be as generalizable for a professional setting.

2.1.3 – N-fold Inspection

The N-fold inspection technique is an idea suggested by (Martin & Tsai, 1990) for improving fault detection in the user requirements document (URD) of mission-critical software e.g. air traffic control or a nuclear reactor. The reason they provide for creating this method is that for regular software an URD that evolves as the project goes on is perfectly normal however for mission-critical software one would desire an URD that remains as static as possible for the duration of the project. Case in point a fault that causes a nuclear meltdown is not fixable after the bug has appeared!

The N-fold inspection works as a regular Fagan inspection, however instead of having one team you have N teams. The reason for N teams instead of one team was based on a pilot study involving ten independent software development teams doing requirements analysis. They found that the faults detected by one team overlapped little with faults found by other teams and that even using only two teams to perform the inspections 13.3 more defects were found on average than a single team (Martin & Tsai, 1990).

Furthermore in a later experiment (Schneider, Martin, & Tsai, 1992) found that a single team found 35% of defects present in a software object while 9 teams found 78% of defects without significant overlap of defects detected by each team although it is noted that there seems to be diminishing returns to adding more teams and the amount of teams one should use is dependent on the importance of the product and the amount of resources available.

2.1.4 – Phased Inspection

Phased inspections were suggested by (Knight & Myers, 1993) and is a new take on software inspection that focuses more on the project of developing a piece of software as a whole as opposed to merely finding defects. Knight and Meyers argue that previously developed methods focus too much purely on defect detection that are typically program breaking and not enough on other aspect of a well-designed program such as i.e. maintainability, portability and reusability.

They further argue that other inspections do not result in specific improvements that a manager or boss can rely on instead they only know that a traditionally performed inspection will result in a general increase in quality, but not specifically where this increase in quality occurs. This gives rise to the term phases. A phase, in this paper, refers to a more targeted inspection that deals with a smaller subset of the project with the goal of improving this specific subset thus giving the people in charge of the project a more tangible knowledge of what area exactly will improve and why. Furthermore the phases should be constructed in such a fashion that they are performed in order and the next phase can assume the existence of properties checked in a previous phase.

They report that when reading about 110 to 875 lines of code per hour their reviewers discovered between 1.5 to 2.75 defects per hour (Knight & Myers, 1993). They compare this result with a similar experiment ran by Russell who reported a defect detection rate of a little more than one per hour at an inspection rate of 150 to 750 lines of code.

They conclude that while the results are good they are not entirely consistent and as such unsatisfactory and suggest that the technique may need further refinement to get consistently good results.

2.1.5 – Inspection without a Meeting

An inspection without a meeting is an inspection where the group part of the review process is cut out either fully or partially.

The reason for having a group phase to start with is Fagan's strong assertion that most of the defects will be discovered during this phase due to synergy gains. In this context "synergy" refers to the assumed positive effect of people working in a group and is backed up by (Fagan, Design and Code Inspections to Reduce Errors in Program Development, 1976) initial reports that most defects were indeed found in the group stage. However, (Johnson & Tjahjono, 1998) points out that this is due to the fact that the Fagan inspection explicitly waits with defect discovery until the group is assembled whereas using other methods where the group stage is used more to collect defects found in the individual stage such synergism does not present itself (Aurum, Petersson, & Wohlin, 2002).

In their experiment (Johnson & Tjahjono, 1998) where they attempted to answer the following question:

"Are there differences in defect detection effectiveness and defect detection cost between subjects who review source code using a synchronous, same-place same-time interactive method and subjects who review source code using an asynchronous, same-place same-time non-interactive method?"

Johnson and Tjahjono found that using the nominal group approach led to more defects detected at a reduced cost and further hypothesize that the approached used by the nominal group members could be subject to further refinement resulting in even more convincing results in favor of non-real groups finding more defects. However, they also note that their findings do not unequivocally support getting rid of group work stating that real groups produce significantly less false positives. They also state that 30% of defects discovered by real groups were due to synergism. They define synergism as a defect discovered in a group setting where no single participant could claim credit for discovering it i.e. the discovered defect was a direct result of collaborative efforts.

Reviewers working in groups, over working alone, also reported a higher sense of satisfaction with the review process which the authors hypothesize is due to an increased amount of certainty in their own work. Someone working alone does not get the added effect of having several people agree with your assessment of whether a detected defect truly is a defect or not.

In summary using groups to discover may not be as cost efficient, but it may have secondary benefits such as i.e. making the members of the project more knowledgeable about the software as a whole and it could also serve as an educational platform for new and inexperienced programmers to interact with and learn from the senior programmers.

2.2 – Static Analysis and Dynamic Testing

It should briefly be mentioned that software inspection is placed under a paradigm referred to as static analysis. In software development static analysis refers to any method of software testing that focuses more on reviewing and analyzing the source code rather than running the program several times with different input to see if it breaks. On the other hand you have dynamic testing which generally does exactly that: feeds various inputs into the program to see if it breaks or if it keeps running no matter what.

It should be noted that neither methods are inherently superior to one another and work in a complementary fashion. Typically any piece of software presented for static analysis has already been through some degree of dynamic testing by the original author to get it to work as well as it does so far. The most comprehensive and successful way of performing a dynamic test is simply performing a brute force test of all possible inputs, however this method is costly and time consuming and often impractical due to the fact that “all possible inputs” is often a very large amount of input! There exist more sophisticated methods of dynamic testing they are, however, beyond the scope of this thesis.

Since software inspection is the focus of this thesis (Fairley, 1978) compiled a list of information obtainable performing static analysis:

- Syntactic errors.
- Number of occurrences of source statements by type.
- Cross-reference maps of identifier usage.
- Analysis of how the identifiers are used in each statement.
- Subroutines and functions called by each routine.
- Uninitialized variables.
- Variables initialized, but unused.
- Unreachable code.
- Deviation from standard coding practices.
- Misuses of global variables, common variables and parameter lists.

2.3 – Reported Benefits of Software Inspection

Lastly I present a list compiled by (McGibbon, Ferens, & Viennau, 2007) in their State of the Art Report which detail the benefits of software review:

- **Reduces Development and Maintenance Costs:** The cost of implementing software improvement methods are heavily outweighed by the cost savings from reduced development costs, and the cost savings resulting from less rework. The major reduction of development costs can be attributed to improved software productivity.
- **Improves Customer Satisfaction:** Typical software development organizations release products with 15% of the defects remaining for the customer to find. No customer is happy

with that many problems. Some software process improvement methods reduce post-release defects to near zero. Improving customer satisfaction is shown to result in repeated customer business and an improved company image.

- **Reduces Cycle Time:** Improvement efforts can reduce typical schedule lengths by 30% to 40%. This may translate to higher profit because it may allow organizations to beat their competition in getting product to the market, it may result in more products purchased earlier than projected, or it may result in schedule related bonuses for early delivery. Combining improved schedules with higher quality - getting better products out sooner - is a winning combination as far as our customers are concerned.
- **Increases Profitability:** The return on investment for software improvement is high. Many organizations have reported a 7:1 return on investment (ROI). This high ROI is achieved by reducing development costs, rework costs, and turnover costs. Product sales increase from higher quality software, penalties turn into bonuses, and repeated business increases. Furthermore, a risk analysis of doing software improvements versus not performing the improvements highly favors performing the improvements.
- **Improves Professional Staff:** Software process improvement improves employee morale, and increases the confidence of developers. It results in less overtime, less crisis, less employee turnover, and an improved competitive edge. The reduction in employee turnover costs and retraining costs could alone pay for the improvement costs.

Chapter 3 – Methodology

The purpose of this chapter is to detail how an experiment is constructed and executed. The material in this chapter and figures 3, 4, 5 and 6 are adapted from (Wohlin, Runeson, Höst, Ohlsson, Regnell, & Wesslén, 2000) and borrows material I have previously written on this subject.

An experiment consists of several phases which are:

- Defining your experiment – *why* is it being done.
- Planning your experiment – *how* is it being done.
- The operation of the experiment which is the preparation and execution of the experiment and the validation of collected data.
- Analysis and interpretation of data follows the operation. In this phase you will analyze the data and draw a conclusion based on your findings.
- The final phase of an experiment is to present your findings in a report.

3.1 – Defining the Experiment

When one first start out creating an experiment there will usually be a general idea of what one wants to accomplish, but there will not be much detail in how to do it. Since performing an experiment involves quite a bit of time and resources one wants to have a solid foundation to avoid wasting the effort involved or, in the worst case scenario, ending up with an experiment that gathered data unrelated to the question one wanted answered.

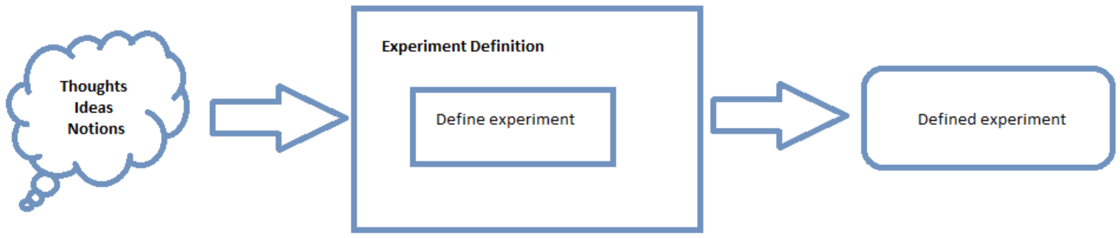


Figure 3 - Experiment definition process.

To prevent this we create a goal template based on GQM which defines the following:

- What is the object of study?
- What is the purpose of the experiment?
- Which effect is being studied?
- In which perspective is the experiment being performed?
- What is the context of the experiment?

The object of study is defined as the entity being studied in the experiment.

The purpose of the experiment defines the intention of the experiment.

The effect is defined as the primary study under effect in the experiment.

The perspective in which the experiment is performed is defined to be the viewpoint of the person or persons interpreting the experiment results for example a project manager or an engineer.

The context of the experiment is defined as the "environment" in which the experiment runs in regards to what kind of personnel (subjects) is involved and which software objects are used.

3.2 – Planning the Experiment

In the process of creating an experiment, the planning part of the progress defines how the experiment will be conducted as opposed to the why it will be conducted laid out by the definition part.

The planning part begins by using the experiment definition as a basis for further progress to construct the next six steps. These six steps are:

- **Context selection**: this stage selects the experiment environment.
- **Hypothesis formulation**: this stage is where the hypothesis is formed as well as the selection of independent and dependent variables.
- **Selection of subjects:** this stage is where the selection of subjects happens.
- **Experiment design**: this stage is based on the hypothesis and variables selected.
- **Instrumentation:** this stage prepares for the practical implementation of the experiment.
- **Validity evaluation:** this stage is intended to check the validity of the experiment and also attempts to find ways beforehand to improve validity.

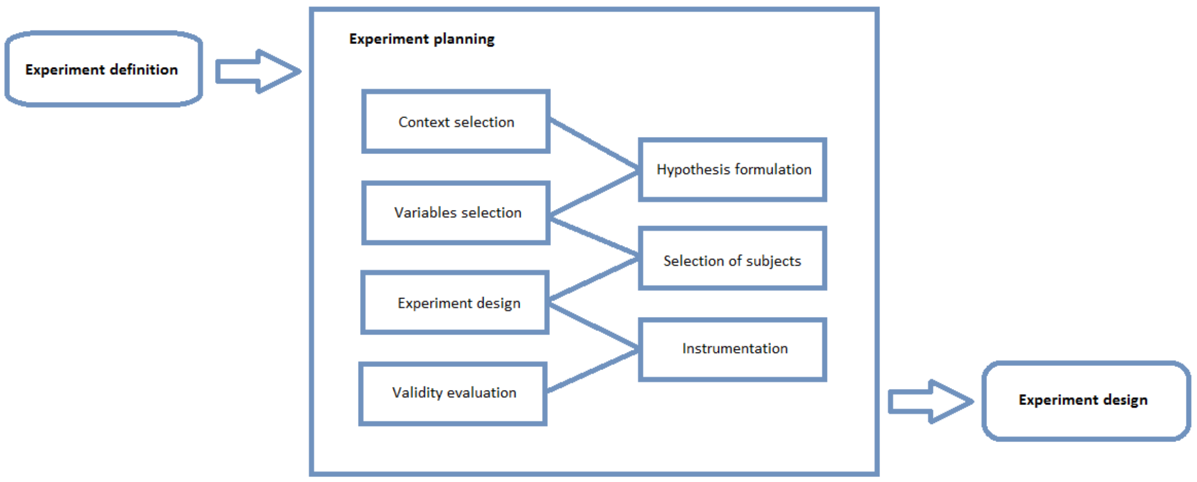


Figure 4 - Experiment planning process.

3.2.1 – Context Selection

In (Wohlin, Runeson, Höst, Ohlsson, Regnell, & Wesslén, 2000) the authors outline the importance of selecting a proper context for an experiment. Ideally one would like to be able to use real life settings: a real company doing a real project using professional staff. However, since any experimental method being tested could possibly be worse than any pre-existing method such an approach would be very risky as performance could well suffer because of it.

An alternative to doing a live (on-line) experiment would be to do run an off-line project in parallel to a real project which would reduce risk, but also increase cost. A cheaper alternative would be to construct a project and have students run it. The pros would be that such a project is cheaper and easier to manage, but the cons would be that students lack the variety and experience of professionals that one would experience in a real life setting. Furthermore the student-run projects usually only involve scaled down versions of problems due to constraints in costs, time and training required which decreases their relevancy to real life problems. Ultimately this comes down to a trade-off between to making the study valid to a specific context or valid to the general software engineering domain.

In conclusion context of an experiment can be characterized according to four dimensions:

- Off-line vs. on-line.
- Students vs. professionals.
- Reduced vs. real problems.
- Specific vs. general.

3.2.2 – Hypothesis Formulation

A hypothesis is a proposition set forth as an explanation for the occurrence of some specified phenomena. In the context of conducting an experiment a hypothesis is a formalized statement about the conclusion the experimenters can draw from the data produced by the experiment. During this phase of experiment design one must form at least two hypotheses the first of which is called the null hypothesis, H_0, and an alternative hypothesis, H_a/H_1/etc.

The null hypothesis states that there are no trends or patterns to be observed in the experiment setting and any observed differences are purely coincidental. A typical null hypothesis would be that the new method on average finds as many faults as the old. Denoted H_0: $\mu_{New} = \mu_{Old}$.

The alternative hypothesis is the hypothesis that indicates that the experiment has revealed a significant difference between the old method and the new for example that the new method is better at detecting defects than the old. Denoted, for example, H_1: $\mu_{Old} < \mu_{New}$ if one expects the new method to perform better.

Once you have devised your hypotheses the experimenter should be aware that there are several statistical tests used to evaluate the outcome of an experiment and that each test involves a risk that the test could either fail to reject a false hypothesis or reject a true hypothesis. The former is referred to as a type-I-error and the latter as a type-II-error.

More formally a type-I-error can be expressed as:

α = P (type-I-error) = P (reject $H_0 | H_0$ true)

and a type-II-error can be expressed as:

β = P (type-II-error) = P (not reject $H_0 | H_0$ false)

When performing hypothesis testing the experimenter wants to reduce the likelihood that the test will reject a true hypothesis. The ability of a test to avoid rejecting a true hypothesis is referred to as the power of a test. This power can formally be expressed as:

Power = $1 - \beta = P(\text{reject } H_0 | H_0 \text{ false})$

An experimenter should choose a test method with as high a power as possible.

3.2.3 – Variable Selection

An experiment consists of independent and dependent variables. The dependent variable is often the result of the experiment and as such can be directly derived from the hypothesis. For example the number of defects detected when comparing to defect detection methods would be a reasonable dependent variable. It should be noted that the choosing of variables can be done in any order, but must be strongly coupled to the hypothesis based on which do data you need to draw a conclusion.

The independent variables are the variables that affect the dependent variable and are the ones an experimenter manipulates to compare a new method to an old one to see if this change results in a change in the results of the new method. The choosing of independent variables can be a difficult task as it is not always apparent which variable or set of variables that generate the largest change in the dependent variable. Independent variables should meet several important criteria:

- The variables should have an effect on the dependent variable and this effect must be controllable.
- The independent variables must be measurable and how to measure it must be known in advance.
- A range must be specified for the independent variables as well as the specific levels for which it will be tested.

3.2.4 – Subjects Selection

As previously mentioned experimenters wants the results for their experiment to be generalizable for as large a population as possible. This requires a careful selection of subjects. One would use subjects as representative of the population as a whole as possible. Such a selection of subjects is referred to as a sample and the size of the sample is referred to as sample size. There are many methods for trying to select a representative sample of the population and such methods can be divided into probability and non-probability sampling methods.

As the name suggests probability sampling methods depends on random selection of subjects. Examples of probability sampling methods are:

- **Simple random sampling:** Create a list of subjects. Pick at random.
- **Systematic sampling:** Create a list of subjects. Pick a subject at random and then every nth person.
- **Stratified random sampling:** The subjects are divided into groups (strata) with a known distribution between groups. Random sampling is then applied to each group.

Examples of non-probability sampling methods are:

- **Convenience sampling:** Pick the nearest and most convenient persons to be selected as subjects.
- **Quota sampling:** Create groupings within a population. Use convenience sampling on the individual groups.

The size of the sample is relevant for our ability to generalize the results. A larger sample creates a more representative subset of the population and is thus more applicable to the generalization of the results however a large sample size requires more time to process and is costly. Generally if the population is diverse a larger sample size is required.

3.2.5 – Experiment Design

The purpose of an experiment is to extract a meaningful conclusion or conclusions and for this purpose to be fulfilled the experiment must be carefully planned and designed. There are many statistical analysis methods available and which method to use is based on the choice made in both which design to use as well as which measurements scales used.

There are three general design principles called randomization, blocking and balancing. These principles entail:

- **Randomization:** The idea here is that all statistical methods used for analyzing data require that observations must be from independent random variables. To make an experiment meet this requirement one makes use of randomization. This is done by randomizing the allocation of objects and subjects as well as applying any tests in random order. This treatment will hopefully average out the effect of any unknown factor or factors as well as select subjects that are representative of the target population.
- **Blocking:** There may be a factor with a known cause and effect which will affect your experiment, but for this experiment one is not interested in this factor and its effects so you can use the technique called blocking. Blocking entails removing or minimizing the effect of this factor from your experiment. For example, if you have a group of subjects where some have experience with the method you want to test and some do not then you can separate the experienced and inexperienced subjects to minimize the effect of prior experience.
- **Balancing:** In an attempt to give each treatment an equal amount of attention one would like to have the same number of subjects testing each treatment.

For a number of different experiment designs there already exist standard templates outlined in (Wohlin, Runeson, Höst, Ohlsson, Regnell, & Wesslén, 2000) which includes:

- One factor with two treatments.
- One factor with more than two treatments.
- Two factors with two treatments.
- More than two factors each with two treatments.

The one of particular interest to this report is one factor with two treatments as the I am planning to conduct an experiment with one factor, amounts of defects discovered, and two treatments, group work versus individual work.

There are two suggested design types for this sort of experiment called *completely randomized design* and *paired comparison design*.

A completely randomized design entails using the same object for all subjects and then randomly assigns each subject to a treatment method. Each subject uses only one treatment on one object. Subjects are evenly distributed between treatment methods.

Subject #	Treatment 1	Treatment 2
1	X	
2		X
3		X
4	X	
5		X
6	X	

Table 1 - An example of treatment distribution for a randomized design.

Suggested statistical tools for data analysis for this design are:

- t-test
- Mann-Whitney

With example hypotheses of:

H_0: $\mu_1 = \mu_2$

H_1: $\mu_1 < \mu_2$, $\mu_1 > \mu_2$

Where the notation μ_i denotes the amount of defects found for detection method i.

A paired comparison design is an attempt to improve the precision of the experiment design by using both treatments on the same object. To minimize the impact of the order in which the treatments are being applied by the subject the order is assigned randomly. For some experiments performing one treatment will provide the subject with too much information about the experiment causing a markedly improved performance with the second treatment. For this reason this design is not always applicable even though it can obtain more precise results.

Analysis of the results of this experiment can be done by checking if the difference between the paired measurements is zero as this would mean that neither treatment is superior or, in case of a non-zero result, imply that one treatment is superior to the other.

Subject #	Treatment 1	Treatment 2
1	2	1
2	1	2
3	2	1
4	2	1
5	1	2
6	1	2

Table 2 - An example of treatment distribution for a paired comparison design.

Example hypotheses for a paired comparison design:

H_0: $\mu_d = 0$

H_1: $\mu_d < 0$, $\mu_d > 0$

Here $d_j = y_{1j} - y_{2j}$ where y_{ij} denotes the j:th measurement of the dependent variable for treatment i and μ_d denotes the mean of the difference.

Suggested statistical tools for data analysis of this design are:

- paired t-test
- Mann-Whitney
- Wilcoxon

3.2.6 – Instrumentation

There are three types of instruments in general for experiments. These are:

- Objects.
- Guidelines.
- Measurements.

Before any experiment can be conducted these three instruments must be defined. An object could be a set of specifications or documents of code. For this experiment, a document with a known set of defects should be prepared for inspection. One could do this either by seeding the defects or using an early version of a real document with previously identified defects.

Guidelines are required to guide the participants of the experiment. Guidelines may include things like process descriptions and checklists. If one were to use several methods of defect detection then a guide on how to perform each of these detections must to be provided to the participants in advance so they would know how to perform them.

Finally, there are the measurements. In this context the measurements to be made from an experiment are conducted by data collection and then performing an analysis on the data one has collected. Data is often collected by doing interviews with the subjects or using a questionnaire. Data is also collected in the form of the results of the main activity of the experiment.

3.2.7 – Validation Evaluation

When designing an experiment it is of vital importance to obtain results that are valid for any larger scale generalization. There are two criteria we would like any experiment results to meet. Firstly the results should be valid for the population the sample was taken from. Secondly the results should be able to be generalized to a different population than the population the test subjects were originally sampled from.

If the results *are* generalizable for a different population the results are said to have adequate validity. Adequate validity does not necessarily imply a most general validity though. For example a company may desire to research a particular method and be able to extract results that can be generalized for the entire company based upon a sample of its workers. However the generalized result may still only hold for this specific company and thus lack general validity.

This need for generalizability has resulted in a number of different classification schemes being proposed to identify threats to the validity of experiment results. In their book (Wohlin, Runeson, Höst, Ohlsson, Regnell, & Wesslén, 2000) they identify four potential threats listed as:

- **Conclusion validity:** This validity concerns itself with whether or not the conclusion is actually based on a real statistical relationship with a given significance or if the experimenters perhaps chose to draw a biased conclusion based on personal beliefs.
- **Internal validity:** This validity concerns itself with whether or not that it is the actual treatment causing the effect that is being observed, a causal relationship, or if there is some uncontrolled or unmeasured factor that is the ultimate cause.
- **Construct validity:** This validity concerns itself with whether or not we have understood the relationship between theory and observation.
- **External validity:** Finally external validity concerns itself with whether or not the results are generalizable. If a causal relationship can be established between cause and effect then one must examine if the results are generalizable to a larger population.

3.3 – Operation of the Experiment

Once we have planned and designed our experiment it is time to carry it out in order to gather data for analysis. It is in the operational phase that the treatment devised in the planning and design is actually tested on subjects. This will often be the first time the experimenters meet and interact with the test subjects which leads to an important realization. Even if the experiment has been carefully planned and the data are analyzed perfectly the results will be invalid if the subjects themselves did not participate seriously in the experiment from the beginning and thus guidelines provided by the field of psychology on how to conduct experiments involving humans are also relevant to the field of software engineering.

There are three operational phases in an experiment and they are:

- **Preparation:** This stage is about where to choose subjects from and preparing forms for subjects to answer etc.
- **Execution:** This stage is where the experiment is performed and data about the different treatments and subjects are collected.
- **Data validation:** This stage is where the experimenters validate their data or, unfortunately, discover that their data is invalid.

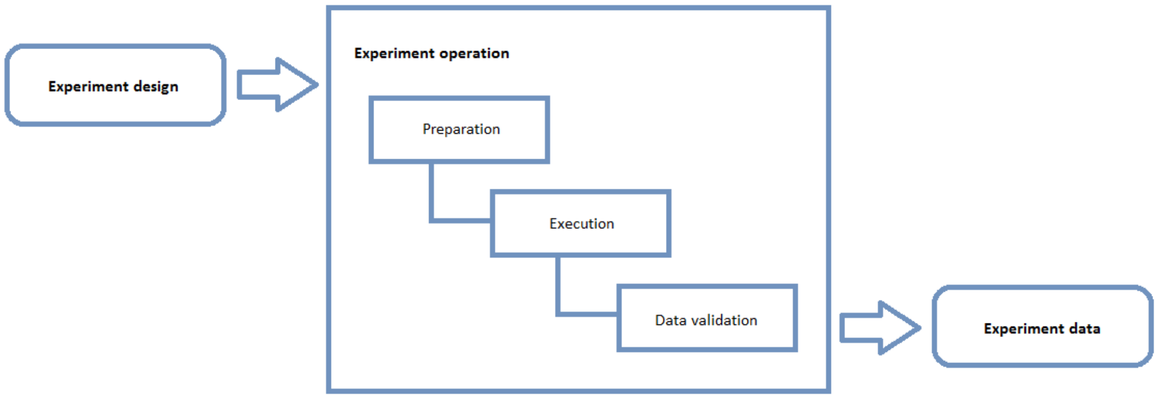


Figure 5 - Experiment operation process.

3.3.1 – Preparation

For the first stage it is important to not only find subjects, but also to ensure that the subjects will be committed to stay and do their proscribed tasks seriously throughout the whole experiment. In addition, the subjects must be properly informed about their tasks and treatment method. It is also important to prepare any materials you will need such as tools and forms.

With regard to subjects selection it is important to find people with relevant experience to what they are being asked to do. For example, if the experiment is based on inspecting Java code it would be a good idea to involve programmers who actually work with or at the very least have a working knowledge of Java. If the experimenters fail to acquire subjects with relevant experience or knowledge this will be a threat to external validity.

As we find the appropriate subjects for our experiment we should start to consider the following aspects about our subjects and their participation:

- **Obtain consent:** The participants must understand and agree to the research objectives. A failure to inform the participants about the experiment may result in the participants executing their roles incorrectly or they may choose to withdraw from the experiment. This may result in an invalidation of the data. In regards to participant withdrawal it should be made clear to every subject that they are free to not continue participation in the experiment at any stage.
- **Sensitive results:** If results obtained from the experiment will reflect on, for example, the performance of a subject it is important to assure participants that this information will be kept confidential. An example of such a scenario would be an experiment conducted to test the skill of programmers where an unfavorable evaluation could be embarrassing and as such it would probably be a good idea to keep this information confidential.
- **Inducements:** As a potential means of increasing motivation a small reward may be offered for participation and completion. However, the reward must not be too big or the experimenter may risk that subjects are only participating to get the reward and not because they seriously wish to participate.
- **Deception:** It may be required that some deception must be used to obtain valid results. An example of this could be a medicinal study where you give one group of subjects the drug one would wish to test and the other group a placebo. It should be noted that deception is

not a desirable method to use and when used the participants should be clearly informed before the experiment is started or as early as possible. Deception should not be used, for example, when explaining to participants that certain sensitive information will be kept confidential and then not follow through on this promise.

In addition to making sure that one has obtained the proper subjects required for the experiment and given them proper instructions on how to perform their tasks, the instruments required by the experiment must also be ready for use. Typical instruments include objects, guidelines, measurement forms and tools required by the experiment. It should be noted that if there is no real need for the experimenter to be able to distinguish between subjects then forms may be filled out anonymously. This will increase the confidence of the subjects that their information is being treated confidentially. It has the drawback, however, of not being able to go back and ask a participant follow up questions in case of particularly interesting feedback.

3.3.2 – Execution

There are several ways to conduct an experiment. In the case of a simple inspection experiment, which is the focus of this paper, it can be properly carried out in one sitting by, for example, gathering all the participants in a meeting room or a similar setting. Two advantages of such an experiment are that since it is performed all at once there is no need to contact participants later on and ask for their results as you have already obtained them. Secondly during such an experiment the experimenter will be present at the meeting and if any participants have questions they can be resolved directly.

For larger experiments this approach is not viable as the experimenter cannot possibly be present to address every facet of the experiment and data collection. However, this concern is outside the scope of this thesis.

After the experiment has been performed, it is time to collect the data. This can be done in several ways where the most common is to let the participants fill out forms (Wohlin, Runeson, Höst, Ohlsson, Regnell, & Wesslén, 2000). Other ways include a manual collection assisted by tools, in interviews or automated by tools.

For collection by forms an advantage for the experimenter is that it does not require active participation and is thus resolved quite quickly. A drawback is that the experimenter cannot gauge the reaction of the subject and reveal inconsistencies or uncertainties in the answers. For collection using interviews an advantage is that the experimenter gets the same advantage as using forms, but gets to participate in the gathering and can probe the participants more carefully and thus hopefully getting better data. The drawback is that this is more time consuming and possibly unnecessary on smaller experiments.

If an experiment is conducted within an actual development project, efforts should be taken to not let the experiment affect the project more than necessary. The reason for this is that the experiment is being conducted in a project setting to measure the effect of a different treatment method in a similar environment. If the experiment then affects the environment this could invalidate the results. However, in the case of the experiment discovering a beneficial effect on project development it could be reasonable for the experiment leader to inform the project leader of these beneficial measures to improve their work. This can have the side effect of motivating experiment participants as they get to see the real value of their work.

3.3.3 – Data Validation

When the dust has settled it is time to validate the data. Common errors include subjects having filled out the forms incorrectly because they misunderstood what the experimenter wanted or because the participants were not being serious. If this is the case the data should be removed from analysis as it is invalid. Another thing to check is whether the experiment was conducted as originally designed. For example the experiment should not have grown in an attempt to gather data it was not originally designed for or the subjects might have executed their treatments in the wrong order. A way to test whether or not the subjects have properly understood their instruction is to give a seminar post-experiment execution and observe whether the subjects are agreeing with the claims made by the researcher or not.

3.4 – Analysis and Interpretation of the Data

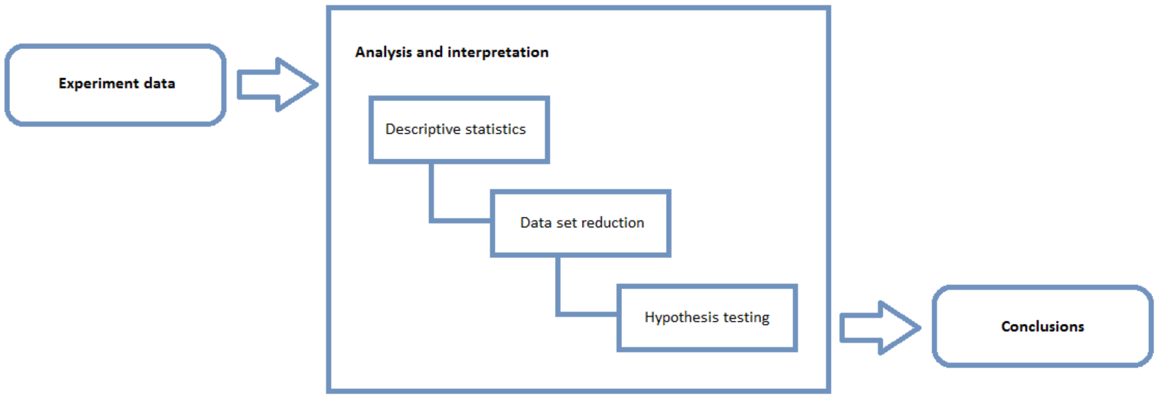


Figure 6 - The analysis and interpretation process.

After having collected the data from the experiment it is time to draw a conclusion based on the data. To do this the data must be interpreted using statistical analysis tools. This is done in the three steps depicted above:

- Descriptive statistics.
- Data set reduction.
- Hypothesis testing.

Descriptive statistics is a way of presenting data visually in a way that is descriptive and allows the reader to make sense of the data hence the name. In the data set reduction step one excludes abnormal or false data points thus reducing the data set. Then the data is analyzed by hypothesis testing where the hypotheses are given a statistical evaluation at a given level of significance.

3.4.1 – Descriptive Statistics

Descriptive statistics deal with the presentation and numerical processing of a data set and is used after the collection of experiment data to describe and present interesting aspects of the data graphically in a way a reader can easily comprehend. The goal of descriptive statistics is to get an overview of how the data set is distributed and is often useful before the hypothesis testing starts to identify abnormal data or false data points referred to as outliers. We would like to use descriptive statistics to get a feel for the following:

Scale Type	Measure of Central tendency	Dispersion	Dependency
Nominal	Mode	Frequency	
Ordinal	Median, percentile	Interval of variation	Spearman corr. coeff. Kendall corr. coeff.
Interval	Mean	Standard deviation, variance, and range	Pearson corr. coeff.
Ratio	Geometric mean	Coefficient of variation	

Table 3 - Meaningful measures for every scale.

These terms are defined to mean:

- **Nominal:** data collected on the nominal scale can only be subjected to the simplest of operations such as equivalence or set membership. Examples on dichotomous values include “male” or “female” distinction when identifying gender or “brown”, “black”, “blonde”, “red” when identifying hair color for non-dichotomous values. This scale only gives us access to the mode measurement.
- **Ordinal:** data collected on the ordinal scale can be subjected to a ranking operation such as race rankings or IQ scores. This allows us to make statements about the median or percentiles, but not the average. For example, it makes sense to refer to a ranking within the 10%-percentile in a race, but it makes no sense to refer to the average ranking of a race participant!
- **Interval:** data collected on the interval scale allows us to make most measurements on the data set. Using the race example above we could acquire the time measurements data set to provide us with more useful data than merely ranking. With access to time measurements we could make statements about the average time it takes to complete the race, make statements about the standard deviation and variance of run times and obtain an estimate for a minimum and maximum amount of time required to complete the race.
- **Ratio:** data collected on the ratio scale is the most precise data. Most measurements in the natural sciences and engineering are made on the ratio scale. Examples of measurements made on the ratio scale includes: mass, length, time, energy, etc. The precise nature of the measurements obtained on the ratio scale allow for all statistical measurements to be used.

It should be noted that these scales are represented in order of power. The nominal scale is the least precise in its measurements and thus we are unable to make use of rigorous statistical measurement tools on this scale. The ratio scale is the most powerful. Every scale can make use of all the measurement tools listed for it or in the less powerful scales.

Measuring the central tendency of a data set is trying to make statements about a “middle” point finding values like mean, median and mode. Often this midpoint is referred to as the average and provides the interpreter with an expectation of the stochastic variable from which the data points in the data set are sampled.

When talking about the central tendency of a data set we assume that there are n data points $x_1 \ldots x_n$ sampled from some stochastic variable the arithmetic mean of which, denoted $\bar{x}$, is calculated using the formula:

$$\bar{x} = \frac{1}{n} * \sum_{i=1}^{n} x_i$$

For example for the data set (2, 2, 3, 7) the mean would be $\bar{x} = 3.5$.

The median value, denoted $\tilde{x}$, refers to the middle value of a data set when the set has been sorted from lowest to highest value. If the data set has an even number of data points then the median value is defined as their mean value. For the data set above the median value would be $\tilde{x} = 2.5$.

When talking about the median it is worth nothing that the median is a special case of the percentile, specifically the 50%-percentile, denoted $x_{50\%}$, meaning that 50% of the sample is below the median value. Generally the percentile is denoted x_p where p% of samples are below this value.

The mode denotes the most commonly incurring sample and would, in the previously mentioned data set, be 2. If there is more than one value as a candidate then the mode can be defined as the middle value of the candidates.

Another way to measure the central tendency is to make use of the geometric mean defined as:

$$\sqrt[n]{\prod_{i=1}^{n} x_i}$$

This mean is well defined if all samples are non-negative.

If the distribution of samples is symmetric the arithmetic mean and median are equal. If the distribution also has one unique maximum then the mean, median and mode are all equal.

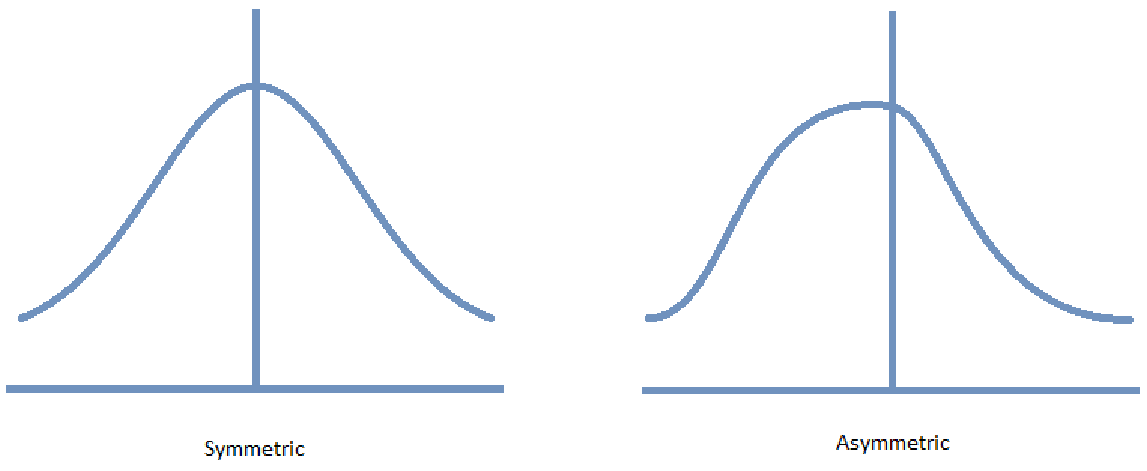


Figure 7 - Examples of symmetric and asymmetric distributions.

In addition to measure the central tendencies of the data there are methods to measure the dispersion of the data points. This gives us information about the spread of the data. For example, if we propose that a new method of defect detection is superior to another we would expect the data points of each method to cluster together distinctively from each other. Methods used for this includes finding the variance, standard deviation and the range of values.

Variance, denoted s^2, is found by calculating:

$$s^2 = \frac{1}{n-1} * \sum_{i=1}^{n} (x_i - \bar{x})^2$$

By subtracting the mean from every data point and then squaring it gives us an impression of how the data is spread. If every data point was equal to the average then there would be no spread and the answer would be 0. Since variance is calculated by squaring the data points this means that the unit of the variance will be the square of the unit used. Because of this it is often preferred to use the standard deviation as it will have the same base unit.

Standard deviation, denoted s, is found by calculating:

$$s = \sqrt{\frac{1}{n-1} * \sum_{i=1}^{n} (x_i - \bar{x})^2}$$

Finding the standard deviation is meaningful for the interval and ratio scales.

Finally the range of a data set is calculated using:

$$range = x_{max} - x_{min}$$

The range value is meaningful for the interval and ratio scales.

Other useful measures for dispersion are:

- **Variation interval:** a value representing the min and max values of the data set denoted (x_{min}, x_{max}). This value is meaningful for the interval and ratio scales.
- **Coefficient of variation:** a value of the dispersion expressed in percentages of the mean calculated as $100 * \frac{s}{\bar{x}}$. This value is meaningful for the ratio scale.
- **Frequency and relative frequency:** the frequency of the data is given by counting the occurrence of each data point. The relative frequency is given by dividing the count by the amount of data points.

Value	Frequency	Relative frequency
1	3	50%
2	1	17%
3	2	33%

Table 4 - Frequencies and relative frequencies for the data set (1, 1, 1, 2, 3, 3).

3.4.2 – Graphical Visualization of Descriptive Statistics

After having calculated the statistical measure one then needs a way to display it. This is done using various forms of graphical visualization. The ones covered in this thesis will be:

- Scatter plot.
- Histogram.
- Cumulative histogram.
- Pie chart.

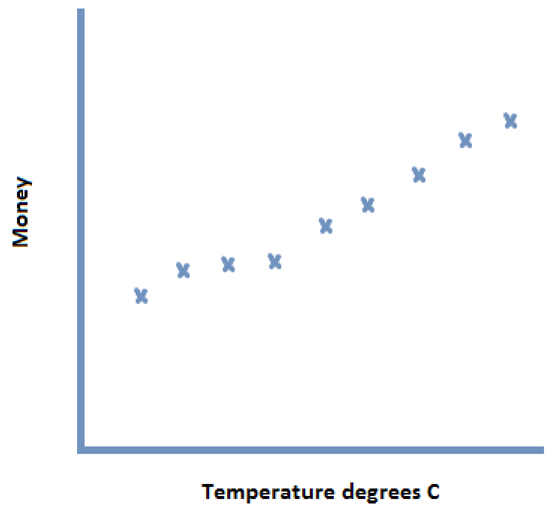


Figure 8 - A scatter plot showing ice cream revenue based on the temperature of the day.

A scatter plot shows the relation of a pair of samples (x_i, y_i) in two dimensions. For example how much money an ice cream seller makes based on the temperature of the day. Useful features of the scatter plot include the ability to see dependencies between two variables and the ability to visualize trends, like ice cream sales increasing on hot days. Scatter plots also make it easier to identify outliers because outlying data points will appear as lone dots apart from the normal cluster of data.

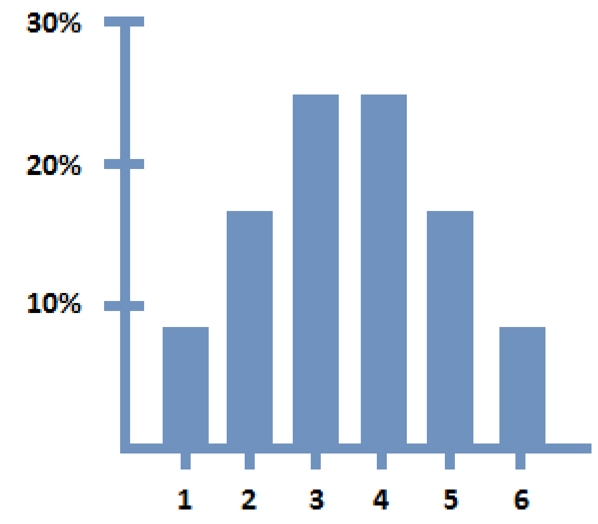


Figure 9 - A histogram for the frequency of dice rolls.

The above figure is an example of a frequency histogram for the data set (1, 2, 2, 3, 3, 3, 4, 4, 4, 5, 5, 6) of dice roll results. This allows us to get a feel for the distribution of the samples of one variable in the data set. For example, here we can easily see that the most commonly rolled values are 3 and 4. One important feature of the histogram is that the basic shape of the samples it provides can be used to observe if the distribution follows a normalized curve. This fact can also be tested formally using statistical tools discussed later.

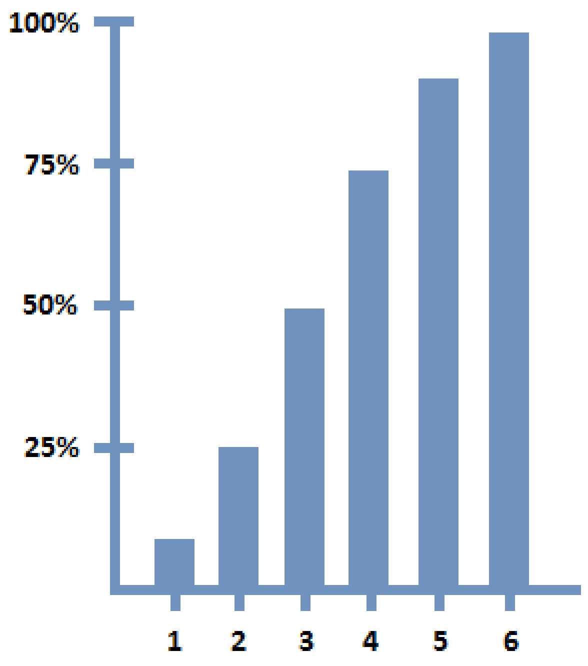


Figure 10 - A cumulative histogram based on figure 9.

The cumulative histogram can be used to get a feel for the probability distribution function of the samples of one variable and lets you observe, for example, the rough range of probabilities for a set of data points.

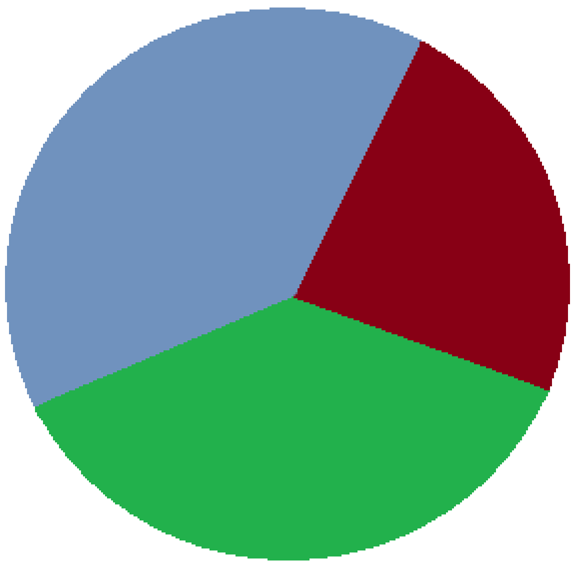


Figure 11 - A pie chart.

Finally, there is the pie chart. The pie chart servers a similar function as the histogram. It provides a good overview of the distribution density of the data. However, where the histogram breaks it down into individual components making it easier to observe if the data has a normal distribution the pie chart does not provide this insight, but instead gives us a clearer view of the entire picture.

3.4.3 – Data Set Reduction

Data set reduction means to eliminate points of data from your complete data set. The point of this is to remove atypical results based on the assumption that they are rare and should not be allowed to influence our conclusion. For example, if one were to evaluate the performance if one were to attempt to measure the quality of phone service provided in Norway one would notice a large decrease in quality around midnight of New Year's Eve. This due to the massive influx of calls made to wish people a happy new year. This observation would have to be scrapped due to being an atypical result.

Furthermore the statistical methods develop for further analysis of data beyond descriptive statistics depend on having inputs that are as representative as possible and will not give proper results if the data set has not been pruned to exclude errors or false data points beforehand.

One way of ridding a data set of outliers is to create a scatter plot as shown earlier. If one or more points are shown to be separated from the others then these points should be considered for removal from the data set. The decision to remove or not should be based on what caused the outlying result. If it was due to rare circumstances that would never happen again or a simple error in the execution of a task it should be removed. However, if there is a chance that a rare event could happen again or the subject turned out to be talented at performing the task then one should consider keeping the data. Every outlier should be judged on a case-by-case basis.

Another way is to observe that the data follows a normal distribution and that allows you to calculate the probability of observing such a result. If the chance is very small then the result can be disregarded.

It should be noted that data set reduction is tied very closely to data validation. Where data validation works in ways such as finding out that a participant worked seriously or performed a task wrong, data set reduction uses statistical methods to analyze the experiment results to identify anomalies.

3.4.4 – Hypothesis Testing

After having performed descriptive statistics tests and reduced the data set it is time to test the hypothesis. The goal of hypothesis testing is to see if it is possible to reject the null hypothesis, H_0, based on the experiment results.

Formally one does this by defining a test unit, denoted t, and a critical area, denoted C, where C is often a range or interval of values within the possible values of t. You can formulate a significance test like this:

- If $t \in C$, reject H_0.
- If $t \ni C$, do not reject H_0.

We could devise an experiment where we could define a null hypothesis as "H_0: observed creature is a human" and make the test unit the amount of legs observed. Discounting accidents or genetic mutations that could cause a human to have a number of legs other than two we could define the critical area as C = {1, 3, 4, 5, 6, …}. The test then becomes if t = 2, do not reject H_0, if otherwise then reject H_0.

It is important to note that observing t = 2 does not confirm the null hypothesis it simply does not provide us with any proof to the contrary. It is thus not rejected and leaves us unable to draw a conclusion.

To test this hypothesis we can make use of statistical tools divided into parametric and non-parametric tests. The differences between the two methods are that for a parametric test some specific assumptions about the parameters involved in the test. It is often assumed that the parameters for a parametric test are normally distributed. It is also required that parametric test parameters be measurable on, at least, an interval scale. If measurements can only be made on the nominal or ordinal scale this generally means that you cannot use a parametric test.

Non-parametric tests are more general in their nature. Non-parametric tests do not necessarily make the same assumptions about the distribution of data as a parametric test and are not as specific in their usage.

The concerns to be noted when selecting which test to use to test your hypothesis are:

- **Applicability:** each test makes an assumption about the input parameters. What are they? These assumptions should be appropriate for your data set.

- **Power:** due to the stricter assumptions applied to parametric tests their power is often greater and thus requires less input parameters making it suitable for smaller experiments who do not acquire as much data granted that the assumptions made are true.

The choice on whether to use a parametric or non-parametric relies on how sure one is of the assumptions made by the parametric tests. If a parametric test is applied with incorrect or false assumptions made then the results are useless. However, the power of a parametric test is also greater than a non-parametric test so the results are more dependable if true. It is worth nothing that simulations have demonstrated that parametric tests are still fairly robust to deviations (Wohlin, Runeson, Höst, Ohlsson, Regnell, & Wesslén, 2000) from the base assumptions as long as the deviations are not too great.

3.5 – Presenting Experiment Results

Finally when an experiment has been performed and the data analyzed it is time to put the results and procedure down in writing so that others may read about your discoveries and attempt to replicate it themselves if they so desire though it should be noted that not all reports are written in such a manner as to be replicable.

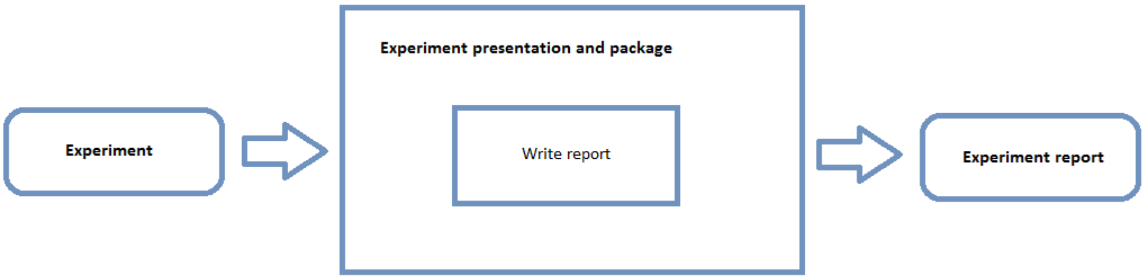


Figure 12 - The presentation process.

The package in which the experiment results are delivered is based on the intentions of the original experiment. It could be written as a paper for a conference, as a report for internal decision making for a company, as a package meant for replication or to be used as educational material.

To make sure that a report contains all the necessary information to reproduce or check the results and the conclusions being drawn from them a report outline has been suggested by (Wohlin, Runeson, Höst, Ohlsson, Regnell, & Wesslén, 2000) and should contain the following:

- **Introduction:** a short introduction about the research area and the objective of the research as well as the motivations for doing the experiment.
- **Problem statement:** the problem statement is in many ways simply a deeper look into the information stated in the introduction. Here one should present how the problem has been observed and why one wants to do something about it. Such a problem statement enlightens the reader as to why the research is being done and it also a suitable time to present related

work done in this area. Along with the problem statement there should be an experiment definition included in this section.

- **Experiment planning:** in this section one includes all the information pertinent to the planning of the experiment including context selection, the hypothesis, variable selection, a method for data analysis, a presentation of the subjects involved and a full description of the experiment design. This section is written so as to provide the reader with a full understanding of how the experiment is done in order to increase their understanding of and confidence in the results.
- **Experiment operation:** in this section one details how the operation is prepared and subsequently executed. Details included here are typically any information that is important to the understanding of how the experiment was carried out and how to replicate it. For example information about the subjects such as whether they were given any information prior to the experiment like whether they had to take lessons prior to the experiment to understand the treatments they were about to execute. How the data was collected is also included in this section as well as data validation procedures taken.
- **Data analysis:** this section presents the analysis of the data where the calculations are described and which assumptions lies behind the selection of an analysis model are provided. Information about sample sizes, significance levels and test applications are also included here. If any actions are taking, such as removing outliers, the rationale is explained here.
- **Interpretation of results:** once the data has been analyzed and presented a conclusion must be drawn. This section provides an interpretation of the results and whether the hypothesis was correct or not or whether the results are inconclusive and cannot confirm nor reject the hypothesis.
- **Discussion and conclusion:** this section deals with the ultimate conclusion based on the interpretation of the data and deals with the outcome of the experiment as a whole as well as any problems encountered along the way, deviations from the plan, etc. If possible the discussion should be related and compared to previous findings by other researchers. This section also typically includes ideas for future work.
- **Appendix:** this section includes all the information that is not necessary to understand the presentation of the experiment. This typically includes the data gathered and information about the instrumentation such as copies of the forms used to obtain data from the subjects and such. If the intention is to create a replicable experiment all material needed to do so would be provided here.

Chapter 4 – Experiment

Based on the research I have encountered so far there seems to be universal agreement that software inspection works. The debate is on how to make the process as efficient and cost-effective as possible. In this regard there is a growing faction that is considering scrapping the group work part of the inspection process as it is costly and has been shown to add a 15-20% overhead (Johnson & Tjahjono, 1998) and does not always produce the desired effect (Lanubile & Visaggio, 1996) of improving the process!

Naturally making the inspection process more costly while not increasing efficiency seems like a horrible idea, however the approach I want to try out is eliminating the individual phase of the inspection process instead, perform a simple inspection experiment where one group of subjects work only as individuals and one group of subjects work only in a group setting and then observe what happens. The theory behind this is that while group work may not always seem to be worth the cost it is also true that sometimes groups perform far better than individuals (Stålhane & Husain Awan, 2005) and it would be interesting to attempt to replicate group work where process gain is positive consistently instead of eliminating it altogether.

Using the process outlined in Chapter 3 I will now describe my experiment.

4.1 – Definition

- **Object of study:** in this experiment the object of study will be the performance of two groups of students. One group working alone and one group working in teams.
- **Purpose of experiment:** the purpose of this experiment is to observe whether one group performs better or worse than the other and also to observe whether the removal of the individual stage of the inspection process has an impact on group work.
- **Effect being studied:** the effect being studied is the rate of defect detection made by groups and individuals.
- **Perspective:** this experiment is being run from the perspective of myself, in the role of a researcher.
- **Context:** the context this experiment is being run in is a student setting. The subjects are 3rd year students with various amounts of programming experience working alone or in groups of three. The software object being studied is a small, three class program available in the Appendixes (A and B).

4.2 – Planning

Here I will provide short rationales for my choices of:

- Context selection.
- Hypothesis formulation.
- Selection of subjects.
- Experiment design.
- Instrumentation.
- Validity evaluations.

4.2.1 – Context

Using the context dimensions defined in chapter 3 my choices end up as shown in Table 5. The reason for this is my current position as a student at the Norwegian University of Science and Technology does not provide me with the resources required to do a full-fledged on-line experiment using professional programmers in an industry setting.

Context Dimensions			
Off-line	X		On-line
Students	X		Professionals
Reduced	X		Real Problems
General	X		Specific

Table 5 - Context Dimensions Selection

4.2.2 – Hypothesis

For this experiment my null hypothesis is that groups that have not prepared individually before the code inspection occurs will perform no worse than nominal groups constructed from the individual participants in the experiment. More formally my null hypothesis, H_0, can be stated as:

H_0: $\mu_{RG} = \mu_{NG}$

Where µ denotes the amount of defects detected and RG/NG denotes real groups and nominal groups.

An alternate hypothesis, H_1, is stated as:

H_1: $\mu_{RG} \geq \mu_{NG}$

4.2.3 – Variable Selection

For this experiment the dependent variable will be the amount of defects discovered by the groups and individuals and the independent variable will be whether a group is a real group or a nominal group constructed by taking the union of individual efforts.

4.2.4 – Subjects

The subjects for this experiment will be 3rd year students from the computer science program of the Norwegian University of Science and Technology. The subjects were picked using convenience sampling i.e. using students from my own university as they are in close proximity and possess relevant education and divided into groups and individuals using simple random sampling. The number of students who attended was 59 thus being enough to obtain statistically significant results.

4.2.5 – Experiment Design

This experiment will have one factor, defects discovered, with two treatments, discovered by real groups or individuals.

The design principle of randomization is addressed by creating the groups randomly from the student subjects and then furthermore the nominal groups are designed using a small piece of Java code that generates a random number sequence from 1 to n where n is the amount of subjects working as individuals. This number sequence is split into groups of three where a number is equal to the result of an individual subject as seen in Appendix D.

The blocking principle is addressed by using a group of subjects that should have roughly the same amount of experience, in this case 3rd year students. Differences in skill amongst the students are compensated for by creating groups randomly which should ensure a fairly equal division of skill. All subjects are working on the same software object which removes any uncertainty whether one software object might have been easier to comprehend and inspect than another.

The balancing principle is addressed by distributing the subjects as evenly as possible between groups and individual work. In this case there were 59 students so there were 30 assigned to real groups and 29 working as individuals.

The time given for performing this experiment was 45 minutes.

By recommendation of my professor the statistical methods going to be used for the analysis of this data will be methods provided by a program called Minitab. Minitab is a statistics package which means that it is a computer program specifically developed for assisting in statistical analysis.

4.2.6 – Instrumentation

The same software object is used for all subjects and can be seen in Appendix A and B with and without seeded defects.

The guidelines provided in this experiment were a short, written explanation of the software object as can be seen in Appendix E. This explanation was provided in Norwegian. Also a short oral presentation was held prior to the execution of the experiment explaining to the students what they were supposed to do with the code they had been given.

Finally, measurements were taken by letting the students hand in their software objects with annotations on them to mark what the student or group of students believed to be a defect along with a filled out questionnaire to give some additional context to the collected data if needed.

Chapter 5 – Data Analysis

This chapter is where the analysis of the data is presented and explained.

5.1 – Statistical Tools

For the analysis the two statistical methods that will primarily be used are the t-test and the two-sample proportion test.

5.1.1 – The t-test

The reason for the use of the t-test is due to the fact that it is suggested by (Wohlin, Runeson, Höst, Ohlsson, Regnell, & Wesslén, 2000) as a way of running a statistical analysis on an experiment design with one factor and two treatments. In this experiment the factor is defects discovered and the treatments are working as an individual or working as a group.

The t-test is a so called parametric test which means it can be performed only when the collected data reaches a certain level of specificity. For example in this experiment where the data is collected as number of defects discovered one can sum up the defects discovered to make statements like "Group 1 discovered x defects" or "The average amount of defects detected for real groups is y" which allows for statistical methods to be used to make statements about means, variances etc. However, if the data had been collected as "Group 1 did well" and "Group 2 did poorly" no such analysis could be done as such data is too nonspecific.

The t-test used in this analysis is provided by Minitab. This program offers three different kinds of t-tests called:

- Paired two-sample for means
- Two-sample assuming equal variances
- Two-sample assuming unequal variances

The paired test is used for when you have two sets of data whose samples are paired. This means if you i.e. test 30 subjects on their ability to perform a test, collect the data, hold a lecture on how to do well on the test and retake the test. You now have a pair of observations for each subject informing you about the effectiveness of the lecture on test scores.

In my design the experiment is only performed once so there are no paired observations to measure so that option is excluded. This leaves equal and unequal variances. Since the base hypothesis is that the two groups of subjects will perform differently unequal variance is assumed and the choice of which t-test to use is thus made.

```
Two-sample T for G-b vs NG1-b

        N  Mean  StDev  SE Mean
G-b    10  6,20   1,14     0,36
NG1-b  10  7,10   1,73     0,55


Difference = mu (G-b) - mu (NG1-b)
Estimate for difference:  -0,900
95% CI for difference:  (-2,294; 0,494)
T-Test of difference = 0 (vs not =): T-Value = -1,38  P-Value = 0,189  DF =
15
```

Figure 13 - A t-test result

Figure 13 shows a sample output of a t-test analysis. This analysis takes the following input:

- **Variable 1 range:** The Minitab cells that contain the variable 1 data.
- **Variable 2 range:** The Minitab cells that contain the variable 2 data.
- **Hypothesized mean difference:** The hypothesized mean difference between the two groups being analyzed, if any.
- **Alpha:** The confidence interval (CI) value. Here the alpha of 0.05 equals a 95% CI.

The formula used for obtaining the t-value, here called T-Value, is:

$$t = \frac{(\bar{x}_1 - \bar{x}_2) - (\mu_1 - \mu_2)}{\hat{\sigma}_{\bar{x}_1 - \bar{x}_2}}$$

Here the $\bar{x}_i$ variables refer to the actual means and the μ_i refers to the hypothesized means which in this case means that $\mu_1 - \mu_2$ should equal 0 as the hypothesized mean difference is that neither treatment is superior to the other and $\hat{\sigma}_{\bar{x}_1 - \bar{x}_2}$ denotes the standard error given by the following formula:

$$\hat{\sigma}_{\bar{x}_1 - \bar{x}_2} = \sqrt{\frac{\hat{\sigma}_1^2}{n_1} + \frac{\hat{\sigma}_1^2}{n_2}}$$

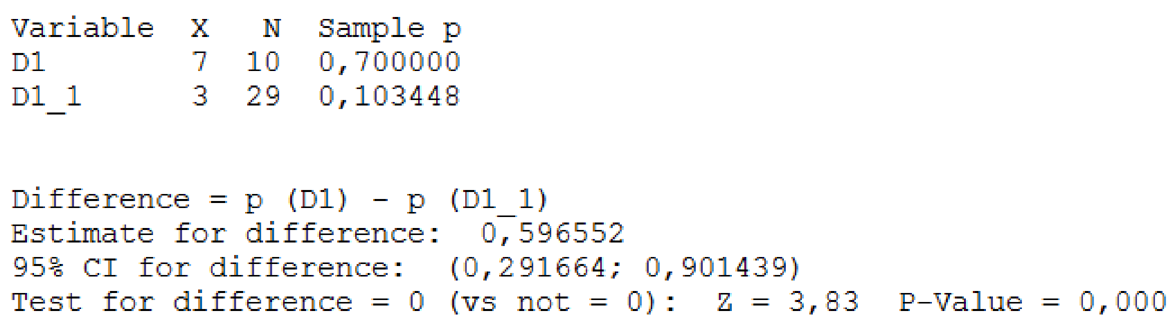
In the case of figure 13 the result of the t-test states that since the p-value is greater than 0.05 it can be concluded that there is no statistically significant difference between the means. One could claim that there is, but there would be a 57.5% risk of being wrong.

5.1.2 – The Two-Sample Proportion Test

While the t-test looks at the two sample populations as a whole and checks for statistically significant differences in the means, the two-sample proportion (TSP) test looks at the differences in the detection rate of each individual defect. This means that the t-test can tell you that the means of the two groups may or may not be different at a statistically significant level, but the TSP test can still provide valuable information about whether real or nominal groups perform better at finding certain defects at a statistically significant level.

The TSP tests whether one factor can affect a variable in two populations. For example two populations of bicyclists: one population wears a helmet and the other does not. Does helmet usage affect the rate of head injuries?

In our case the TSP test can be used to check whether working in a group or as an individual can affect the rate of defect detection in two populations of 3rd year computer science students. The TSP test is provided in the statistical analysis software tool Minitab which has been used in this analysis.



```
Variable   X   N  Sample p
D1         7  10  0,700000
D1_1       3  29  0,103448


Difference = p (D1) - p (D1_1)
Estimate for difference:  0,596552
95% CI for difference:  (0,291664; 0,901439)
Test for difference = 0 (vs not = 0):  Z = 3,83  P-Value = 0,000
```

Figure 14 – A TSP test partial output

Figure 14 shows a sample output of a TSP test. This analysis takes the following input:

- **First sample:** the amount of trials and the amount of successes. In this case the number of groups total and the number of groups who found the defect being analyzed.
- **Second sample:** same as first sample except for the other population. In this case the amount of individuals total and the amount of individuals who found the defect.
- **Confidence level:** inputting your desired CI.
- **Test difference:** input to check whether there is a difference in the populations, 0.0 being the input to test for assumed equal populations.

The following formula is used to obtain the confidence interval:

$$(\hat{p}_1 - \hat{p}_2) \mp z \times \sqrt{\frac{\hat{p}_1(1-\hat{p}_1)}{n} + \frac{\hat{p}_2(1-\hat{p}_2)}{m}}$$

Where $(\hat{p}_1 - \hat{p}_2)$ is equal to:

$$\hat{p}_1 - \hat{p}_2 = \frac{x_1}{n} - \frac{x_2}{m}$$

And z is equal to:

$$z = \frac{\hat{p}_1 - \hat{p}_2}{\sqrt{\frac{\hat{p}_1(1-\hat{p}_1)}{n} + \frac{\hat{p}_2(1-\hat{p}_2)}{m}}}$$

Where x_i refers to defects discovered and n and m refer to the amount of members in their respective populations.

In figure 13 we can see that "Test for difference = 0" means that we are testing to see if real groups and individuals are both equally effective at discovering defect D1. The p value for this claim is 0.000 so this claim is rejected and there is strong evidence that real groups are better at discovering defect D1.

5.2 – The Analysis

The raw data for the analyses done in this sub-section are available in Appendix D.

To create a population to compare the results of the real groups I wrote a short Java program that generated a random sequence of numbers from 1-29 where a number is equal to an individual i.e. 8 would be equal to the results of individual #8. The four sets of nominal groups were generated from the following number sequences:

- 1, 28, 17, 25, 13, 23, 29, 11, 21, 6, 4, 15, 16, 22, 2, 8, 24, 14, 27, 12, 26, 9, 19, 3, 7, 20, 18, 10, 5
- 2, 24, 6, 21, 29, 4, 28, 5, 10, 13, 27, 19, 26, 15, 22, 7, 25, 18, 3, 14, 11, 20, 16, 17, 8, 1, 12, 23, 9
- 23, 9, 29, 10, 22, 13, 5, 26, 18, 4, 3, 8, 17, 14, 11, 24, 19, 15, 28, 25, 1, 12, 21, 2, 6, 16, 27, 20, 7
- 14, 10, 3, 19, 2, 4, 13, 7, 12, 16, 15, 6, 26, 29, 8, 1, 9, 27, 21, 25, 11, 28, 24, 18, 20, 17, 22, 5, 23

I chose to make four sets of nominal groups simply to increase statistical accuracy seeing as the performance of a nominal group is left to the luck of the draw based on which members compose the different groups in a set.

For each set of nominal groups a t-test will be performed. The test checks whether or not the difference in average amount of defects detected is significant.

5.2.1 – Real Groups vs Individuals

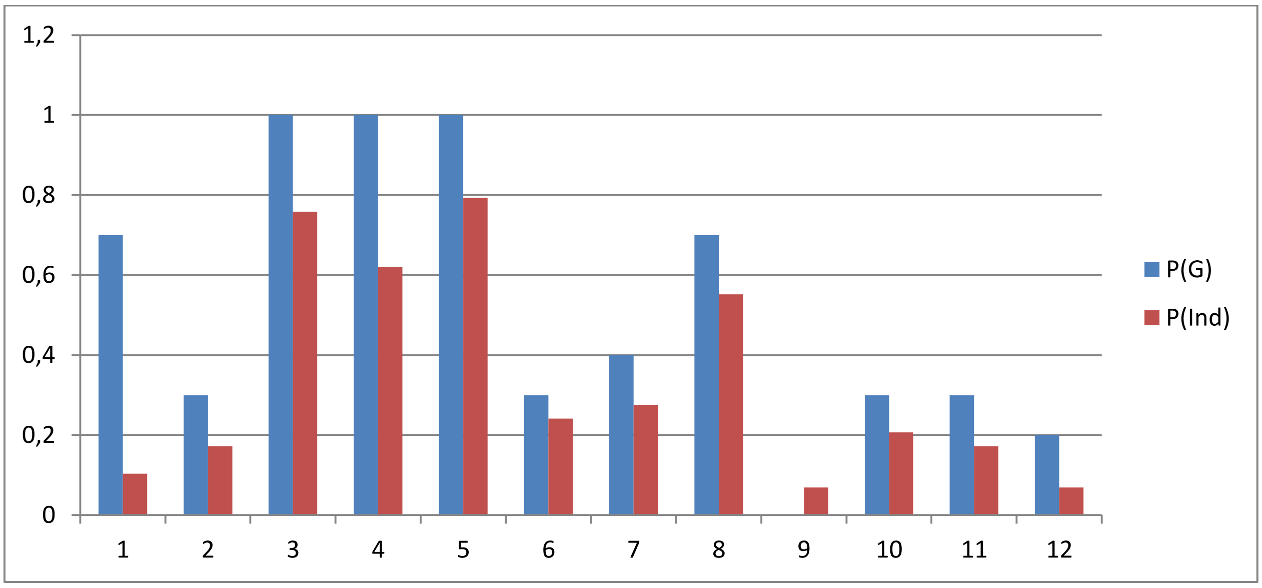


Figure 15 - A column chart demonstrating the probabilities of defect discovery for real groups and individuals

Figure 15 shows the probabilities for a real group or an individual to find defect x. “G” indicates real groups and “NG” indicates nominal groups.

ID	N	Mean	StDev	SE Mean	T-Val	P-Val
RG	10	6.2	1.14	0.36	4.16	0.000
Ind	29	4.03	2.03	0.38		

Table 6 - t-test results for defects discovered by real groups compared to individuals

As we can see from the p-value in table 6 there is a statistical basis for claiming that real groups outperform the efforts of individuals at finding defects. This is to be expected since the combined efforts of three subjects are expected to outperform one.

ID	P(G) – P(Ind)	p-val
D1	0.60	0.00
D2	0.13	0.43
D3	0.24	0.00
D4	038	0.00
D5	0.21	0.01
D6	0.06	0.72
D7	0.12	0.48
D8	0.15	0.39
D9	-0.07	0.14
D10	0.09	0.57
D11	0.13	0.43
D12	0.13	0.33

Table 7 - A TSP test result comparing real group and individual performance

The results of the TSP test are too comprehensive to detail here, but table 7 shows a condensed summary. A p-val less than 0.05 indicates that there has been discovered a statistically significant relationship of one group of subjects outperforming another at discovering defect x.

In plain English this read as:

- Real groups are better than individuals at detecting defects D1, D3, D4 and D5.

This result is discussed in the data interpretation section.

5.2.2 – Real Groups vs Nominal Group Set #1

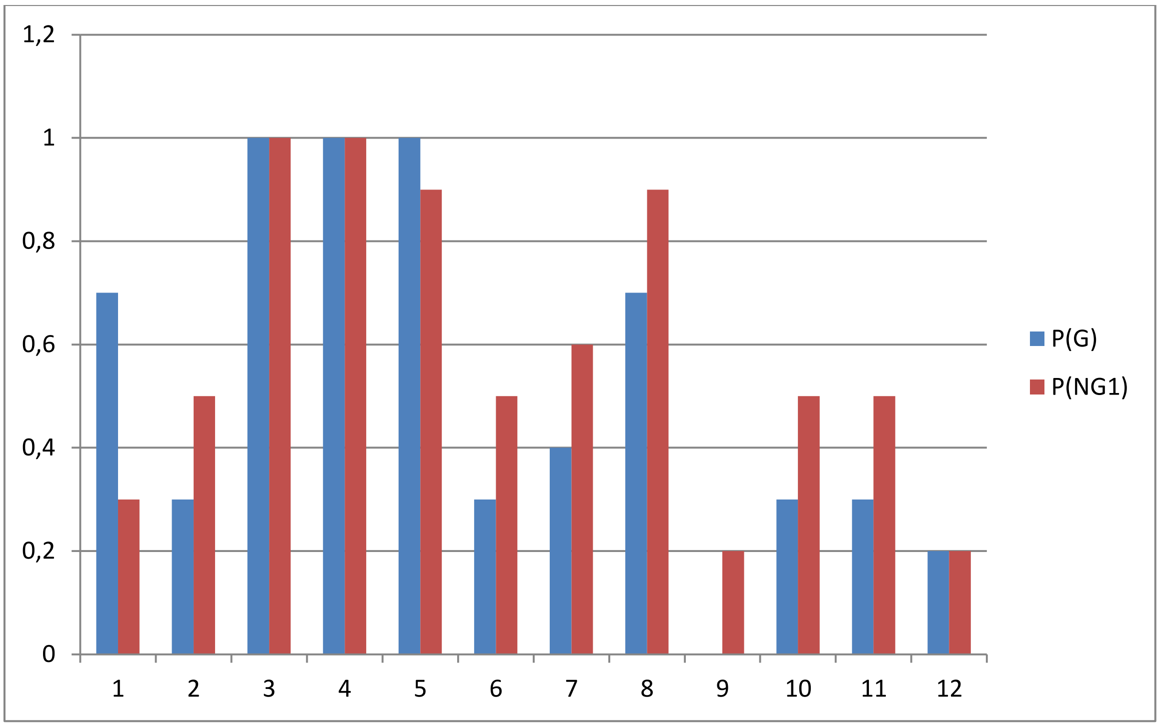


Figure 16 - A column chart demonstrating the probabilities of defect discovery for real groups and NG #1

Figure 16 shows the probabilities for a real group or a nominal group to find defect x. “G” indicates real groups and “NG” indicates nominal groups.

ID	N	Mean	StDev	SE Mean	T-Val	P-Val
RG	10	6.2	1.14	0.36	-1.38	0.189
NG #1	10	7.1	1.73	0.55		

Table 8 - t-test results for defects discovered by real groups compared to NG #1

Based on the p-value here, also used as the example earlier, the hypothesis that the means are different on a statistically significant level is rejected. There is no basis to suggest that either group of subjects outperform the other at a group level.

ID	P(G) – P(NG)	p-val	P(Ind) – P(NG)	p-val
D1	0.40	0.05	-0.20	0.21
D2	-0.20	0.35	-0.33	0.06
D3	0.00	-	-0.24	0.00
D4	0.00	-	-0.38	0.00
D5	0.10	0.29	-0.11	0.38
D6	-0.20	0.35	-0.26	0.14
D7	-0.20	0.36	-0.32	0.07
D8	-0.20	0.25	-0.35	0.01
D9	-0.20	0.11	-0.13	0.33
D10	-0.20	0.35	-0.29	0.09
D11	-0.20	0.35	-0.33	0.06
D12	0.00	1.00	-0.13	0.33

Table 9 - TSP test results comparing real groups and NG #1

The TSP test result for real groups vs NG #1 reads as:

- Real groups are better than nominal groups at detecting defects D1.
- Nominal groups are better than individuals at detecting defects D3, D4 and D8.

The TSP test turned out to output the exact same result for all four runs, so only this one run will be written down.

5.2.3 – Real Groups vs Nominal Group Set #2

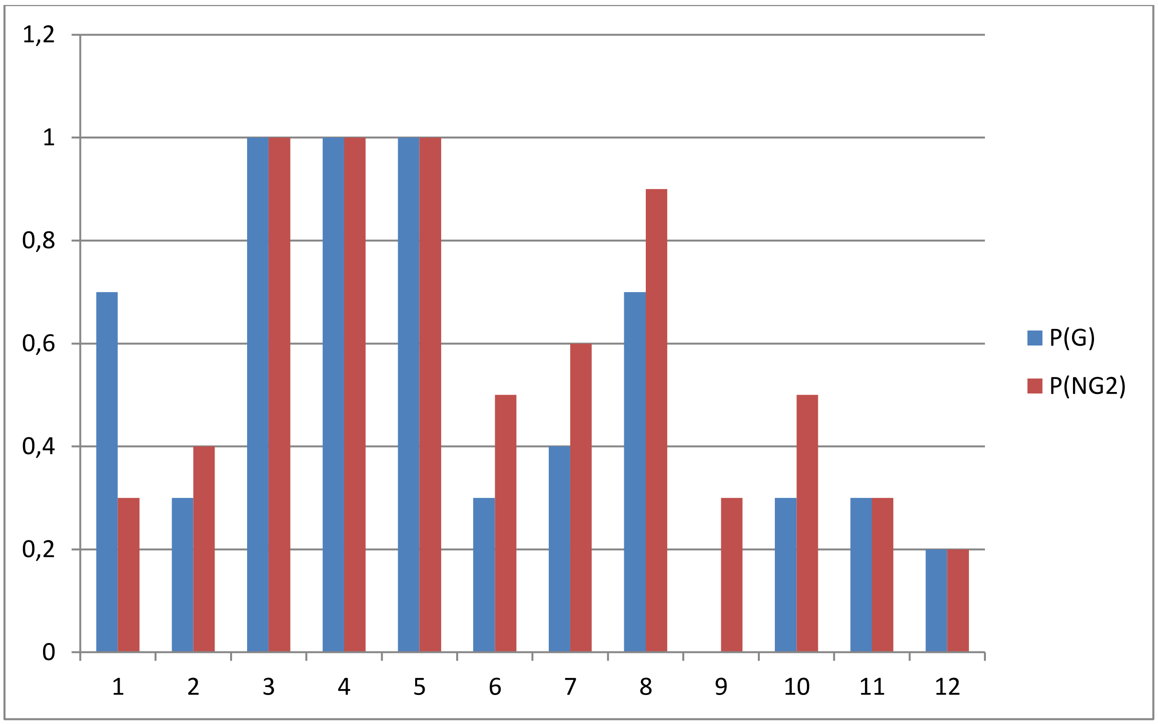


Figure 17 - A column chart demonstrating the probabilities of defect discovery for real groups and NG #2

ID	N	Mean	StDev	SE Mean	T-Val	P-Val
RG	10	6.20	1.14	0.36	-1.18	0.258
NG #2	10	7.00	1.83	0.58		

Table 10 - t-test results for defects discovered by real groups compared to NG #2

Based on the p-value here the hypothesis that the means are different on a statistically significant level is rejected. There is no basis to suggest that either group of subjects outperform the other at a group level.

5.2.4 – Real Groups vs Nominal Group Set #3

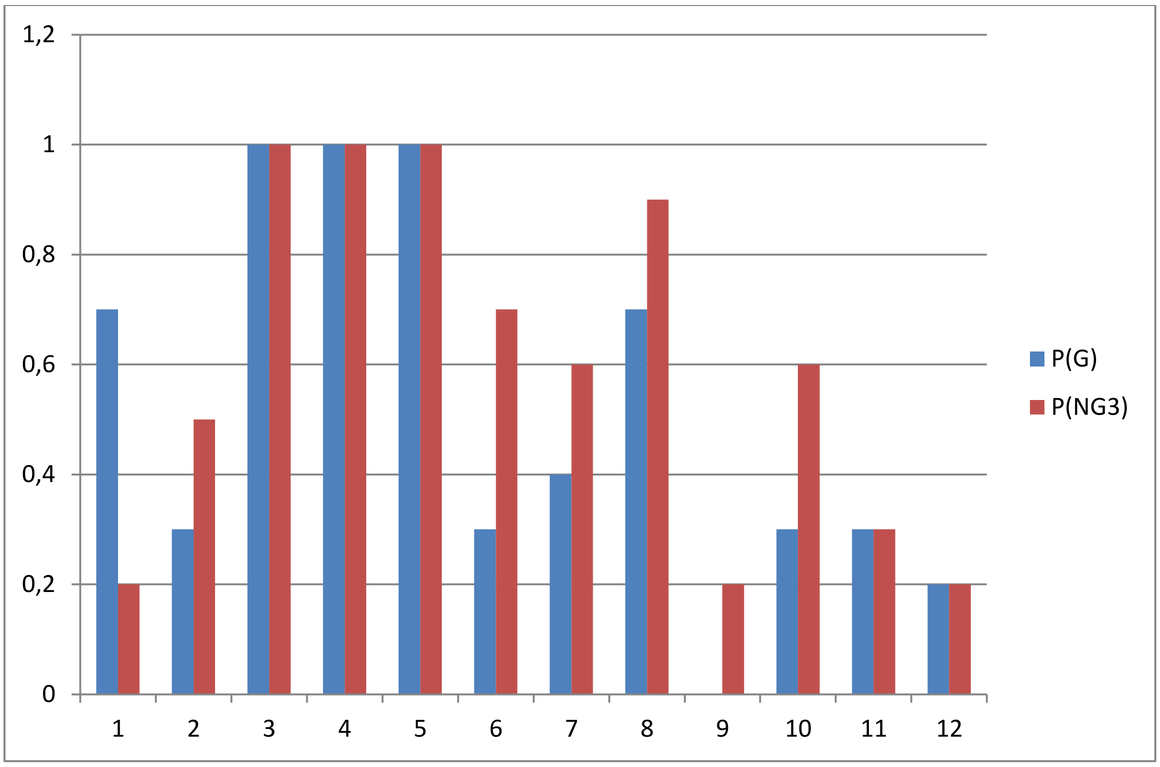


Figure 18 - A column chart demonstrating the probabilities of defect discovery for real groups and NG #3

ID	N	Mean	StDev	SE Mean	T-Val	P-Val
RG	10	6.20	1.14	0.36	-1.38	0.189
NG #2	10	7.20	1.99	0.63		

Table 11 - t-test results for defects discovered by real groups compared to NG #3

Based on the p-value here the hypothesis that the means are different on a statistically significant level is rejected. There is no basis to suggest that either group of subjects outperform the other at a group level.

5.2.5 – Real Groups vs Nominal Group Set #4

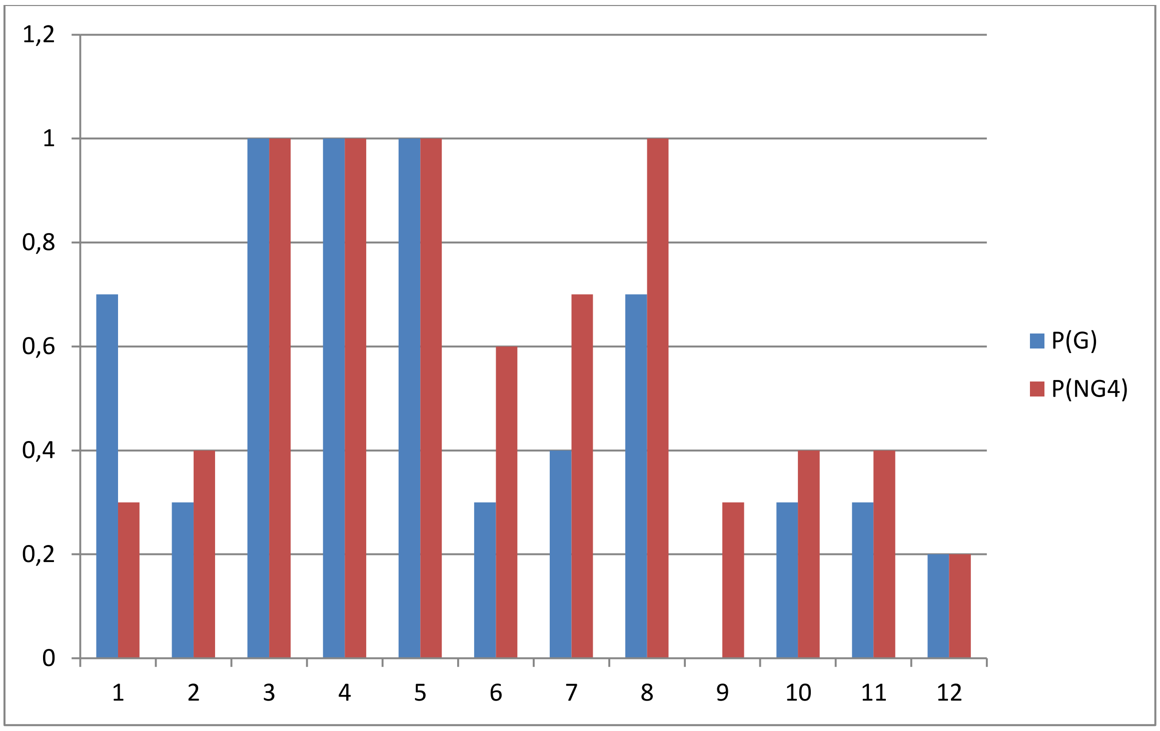


Figure 19 - A column chart demonstrating the probabilities of defect discovery for real groups and NG #4

ID	N	Mean	StDev	SE Mean	T-Val	P-Val
RG	10	6.20	1.14	0.36	-1.58	0.137
NG #2	10	7.30	1.89	0.60		

Table 12 - t-test results for defects discovered by real groups compared to NG #4

Based on the p-value here the hypothesis that the means are different on a statistically significant level is rejected. There is no basis to suggest that either group of subjects outperform the other at a group level.

5.2.6 – Comparison of Results

For the comparison of the results obtained so far we will make use of a method called the Kolmogorov–Smirnov test which can be used to compare the probability distributions given by two samples. The test measures if the largest observed difference, D, in the probability distributions is greater than a number given by the following function:

$$D_{n_1,n_2} > c(\alpha)\sqrt{\frac{n_1 + n_2}{n_1 n_2}}$$

Where α denotes the desired signifiance level and whose values are determined by the following table:

α	0.10	0.05	0.025	0.01	0.005	0.001
c(α)	1.22	1.36	1.48	1.63	1.73	1.95

Table 13 - Values for c(α) given α

To get workable data it is formatted as shown in table 14.

Group #	G	NG #1	NG #2	NG #3	NG #4	Sum NG/4	G	diff
1	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
2	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
3	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
4	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,00
5	0,17	0,17	0,17	0,25	0,08	0,17	0,17	0,00
6	0,50	0,33	0,33	0,25	0,33	0,32	0,50	0,18
7	0,75	0,33	0,50	0,42	0,42	0,42	0,75	0,33
8	0,83	0,67	0,67	0,58	0,67	0,65	0,83	0,18
9	0,83	0,83	0,75	0,75	0,67	0,75	0,83	0,08
10	0,83	0,83	0,83	0,83	0,83	0,83	0,83	0,00

Table 14 - A cumulative, normalized histogram of defect discovery by real and nominal groups

Using the data in table 14 we obtain the following chart:

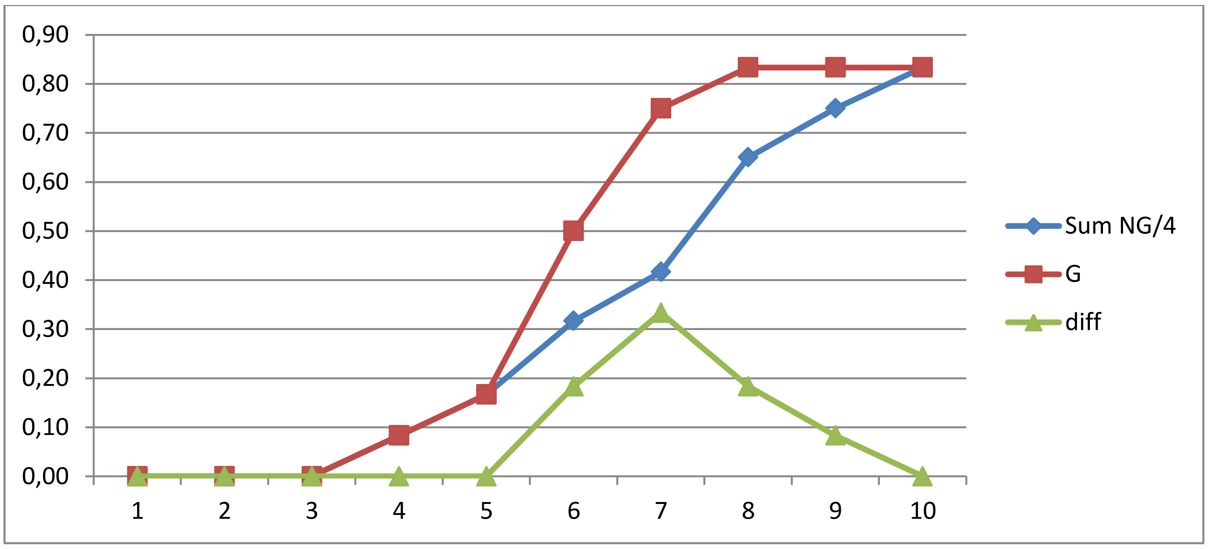


Figure 20 –A chart showing the probability distributions for the averaged out nominal groups and the real groups. The green line displays the difference between the two.

Using the previously listed formula we obtain that the observed difference must be greater than 0.61, however in this case the greatest observed difference is 0.33 so we cannot reject the hypothesis that the probability distributions are the same at a statistically significant level.

5.3 – Data Interpretation

Based on the findings of the previous section it is shown that working in groups or collecting the results of individuals and taking the union of their discovered defects has no significant effect on the amount of defects discovered which answers RQs 1 and 2.

However according to the TSP test it is shown that real groups perform significantly better at detecting the first defect with a 70% detection rate for real groups against a little over 10% for individuals. It is my conjecture that this is due to the fact that both groups of subjects, real and nominal, have never seen the code before the experiment. The subjects working as individuals may be prone to “skimming” the earlier parts of the source code attacking the meat of the code at once. Subjects in a group, however, would have to make sure all group members are following and understand the code reviewed so far thus possibly making the review process a bit slower, but also more thorough.

It is also interesting to note that both groups of subjects perform poorly at discovering the defects D9-D12. All of these defects are involved with the File object of the Java API. The reason for this poor performance may be due to the fact that knowledge of the API of a programming language is often provided by the integrated development environment (IDE) or the internet so when working by hand it would be hard to know whether a piece of code is valid due to lacking knowledge about how the object works.

Defects D2 and D6 were also hard to detect based on the results with a 30% detection rate for both defects for real groups and 15.5% and 19.8% respectively for nominal groups. These defects are of the type “extra” statements meaning that they are valid code, but they are never used and as such are superfluous. Reasons for this might be that the students only had 45 minutes available for defect detection and finding defects that break the program would likely take priority when searching for defects.

Ultimately the findings of this analysis cannot reject my initial null hypothesis and it is concluded that for the sake of finding defects neither group of subjects outperform the other.

Chapter 6 – Conclusions and Further Work

This chapter makes conclusions based on the initial research questions (RQs).

6.1 – Conclusions

6.1.1 – RQ1

"Are real groups better than nominal groups when it comes to identifying defects using code reviews?"

The data indicates that the defect detection rate of real and nominal groups are not different on a significant level. However, it should be noted that one major argument against group work is that it is costly (Votta Jr., 1993) and if the preparation stage of the traditional inspection process can be removed without a penalty to defect detection rates then at least some of that cost is reduced making meetings a more viable option.

Meeting based reviews have also been shown to reduce false positives and increase reviewer satisfaction of their own work (Johnson & Tjahjono, 1998). In addition group settings within a workplace provides a natural way of educating new employees as well as making members of the project more aware of the current state of the source code increasing team wide knowledge.

In conclusion while this experiment shows similar findings to other experiments I do not think group work should be completely written off as a way of finding defects in software review.

6.1.2 – RQ2

"Are real groups better than individuals when it comes to identifying defects using code reviews?"

Based on the results of the previous chapter real groups are definitely better at discovering defects than individual subjects. However, as previously mentioned, this was to be expected due to three subjects being considered better than one, on average. What may be interesting to note, though, is which defects real groups were considered to be better at discovering than individual subjects. This is discussed further in the next sub-section which addresses this scenario of one method outperforming the other.

6.1.3 – RQ3

"If one type of review is better than the others, is this uniformly true or will the conclusion differ depending on defect types? "

In the previous chapter it was observed that real and nominal groups perform better than individuals at finding certain defects. One obvious reason for this observation is that for both real and nominal groups the results reflect the findings of three persons while individual results do not.

The other observed differences between real and nominal groups could possibly be explained by the fact that a group process could function in such a way that individual members cannot rush ahead of the others in reading the code, but every member has to be able to follow the collective train of thought for their group leading to a naturally more normalized process whereas the review process for individuals would vary based on the reviewers habits.

In conclusion I will say that the observed differences in the detection of specific detects do not vary wildly between real and nominal groups and as such is in need of further research involving itself specifically with more advanced defect categories and how these categories affect defect detection rates.

6.1.4 – Final Remarks

It should be noted that while the results found in this experiment are potentially interesting the experiment was performed by an all student group of subjects and it is thus hard to know how much of these results can be generalized for a professional setting as well as this being a single study. More research would be needed to say anything definitive.

6.2 – Further Work

There has been done a large amount of research on the subject of software inspection considering a wide variety of factors and how they may affect the review process which has provided ideas for further work.

Fagan's original inspection has the original author of the code as a member of the inspection group. What is the effect of having the author in or out of the group?

Approaches other than ad-hoc have been used to aid the inspection process. Could the use of checklists or scenarios improve efficiency using this thesis' modified version of the inspection process?

The students for this experiment were given 45 minutes to discover defects. What if more time was given? What if less time was given?

We would also suggest looking into secondary effects of group meetings such as testing to see how much of an effect group meetings have on improving understanding of the project and how it could relate to training new workers. Since cost is a huge factor in deciding which treatment to use in professional settings it is important to know the full ramifications of group work as the process may be more costly as a whole, it could also serve to reduce costs in other areas.

Appendix A – Source Code without Defects

A1 - Class 1: EncryptionServer

```
import java.net.*;

import java.io.*;

/**

* EncryptionServer loops forever, listening for client connection requests

* on a ServerSocket. When a request comes in, EncryptionServer accepts the

* connection, creates a new EncryptionServerThread object to process it,

* hands it the socket returned from accept, and starts the thread. Then

* the server goes back to listening for connection requests. The

* EncryptionServerThread object communicates to the client by reading from

* and writing to the socket

*/

public class EncryptionServer {

        public static void main(String[] args) throws IOException {

        // connection socket

        ServerSocket serverSocket = null;

        // keep running the server

        boolean listening = true;

        try {

                // open the socket

                System.out.println("Up and running");

                serverSocket = new ServerSocket(4444);

        } catch (IOException e) {

                System.err.println("Could not listen on port: 4444.");

                System.exit(-1);

        }

        // keep running the server

        while (listening) new EncryptionServerThread(serverSocket.accept()).start();

                serverSocket.close();

        }

}
```

A2 - Class 2: EncryptionClient

```java
import java.io.*;

import java.net.*;

/**

* Sends the user’s input to the EncryptionServer, which returns the encrypted version

* of the input.

*/

public class EncryptionClient {

        public static void main(String[] args) throws IOException {

        // connection to the server

        Socket socket = null;

        // write to the server

        PrintWriter out = null;

        // read from the server

        BufferedReader in = null;

        // text sent from the server

        String fromServer;

        // text given from the user via command line

        String fromUser;


        try {

                // open up the connection on the given port

                socket = new Socket("localhost", 4444);

                // prepare input/output streams to the server

                out = new PrintWriter(socket.getOutputStream(), true);

                in = new BufferedReader(new

                InputStreamReader(socket.getInputStream()));

        } catch (UnknownHostException e) {

                System.err.println("Don’t know about host: encyptionHost");

                System.exit(1);

        }


        // get input from the user via command line, until the server sends "Bye"

        BufferedReader stdIn =
        new BufferedReader(new InputStreamReader(System.in));

        while ((fromServer = in.readLine()) != null) {

                System.out.println("Server: " + fromServer);

                if (fromServer.equals("Bye")) break;

                fromUser = stdIn.readLine();

                if (fromUser != null) {

                        System.out.println("Client: " + fromUser);

                out.println(fromUser);

                }

        }


        // cleanup

        out.close();

        in.close();

        stdIn.close();

        socket.close();

        }

}
```

A3 - Class 3: EncryptionServerThread

```java
// separate thread for each client

class EncryptionServerThread extends Thread {

        // connection to the client

        private Socket socket = null;


        // constructor

        public EncryptionServerThread(Socket socket) {

                this.socket = socket;

        }


        // keep talking to the client, until the SecretEncryptor sends "Bye" to us

        public void run() {

                System.out.println("Got a new client");

                try {

                        PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

                        BufferedReader in = new BufferedReader(

                        new InputStreamReader(socket.getInputStream()));

                        String inputLine, outputLine;

                        out.println("Welcome to the encryption server");

                        while ((inputLine = in.readLine()) != null) {

                                outputLine = "output>>" + SecretEncryptor.encrypt(inputLine
);

                                out.println(outputLine);

                                if (outputLine.equals("Bye")) break;

                        }

                        out.close();

                        in.close();

                        socket.close();

                } catch (IOException e) {

                        e.printStackTrace();

                }

        }

}
```

Appendix B – Source Code with Defects

B1 - Class 1: EncryptionClient

```
import java.io.*;

import java.net.*;


/**

* Sends the user’s input to the EncryptionServer, which returns the encrypted version

* of the input.

*/

public class EncryptionClient {


        // the main method for this class

        public void main(String[] args) throws IOException {

                boolean validInput = false;

                // connection to the server

                Socket socket = null;

                // write to the server

                PrintWriter out = null;

                // read from the server

                BufferedReader in = null;

                // text sent from the server

                String fromServer;

                // text given from the user via command line

                String fromUser;


                try {

                        // open up the connection on the given port 4444

                        socket = new Socket("localhost", 4445);

                        // prepare input/output streams to the server

                        out = new PrintWriter(socket.getOutputStream(), true);

                        in = new BufferedReader(

                               new InputStreamReader(socket.getInputStream()));

                } catch (UnknownHostException e) {
```

```
                System.err.println("Don’t know about host: encyptionHost");
                System.exit(1);
            }
            // get input from the user via command line, until the server sends "Bye"
            BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in))
;
            while ((fromServer = in.readLine()) != null) {
                System.out.println("Server: " + fromServer);
                if (fromServer.equals("Bye")) break;
                fromUser = stdIn.readLine();

                if (fromUser != null) {
                    System.out.println("Client: " + fromUser);
                    out.println(fromUser);
                }
            }
            // cleanup
            // out.close();
            in.close();
            stdIn.close();
            socket.close();
        }
}
```

B2 - Class 2: EncryptionServer

```
import java.net.*;

import java.io.*;

/**

* EncryptionServer loops forever, listening for client connection requests

* on a ServerSocket. When a request comes in, EncryptionServer accepts the

* connection, creates a new EncryptionServerThread object to process it,

* hands it the socket returned from accept, and starts the thread. Then

* the server goes back to listening for connection requests. The

* EncryptionServerThread object communicates to the client by reading from

* and writing to the socket

*/

public class EncryptionServer {

        public static void main(String[] args) throws IOException {

                // connection socket

                ServerSocket serverSocket = null;


                // keep running the server

                boolean listening = false;

                try {

                        // open the socket

                        serverSocket = new ServerSocket(4444);

                } catch (IOException e) {

                        System.err.println("Could not listen on port: 4444.");

                        System.exit(-1);

                }


                // keep running the server

                while (listening) {

                        new EncryptionServerThread(serverSocket.accept()).start();

                }

                serverSocket.close();

        }

}
```

B3 - Class 3: EncryptionServerThread

```
// separate thread for each client

class EncryptionServerThread extends Thread {

        // connection to the client

        private Socket socket = null;


        // constructor

        public EncryptionServerThread(Socket socket) {

                this.socket = socket;

        }


        // keep talking to the client, until the SecretEncryptor sends "Bye" to us

        public void run() {

                System.out.println("Got a new client");

                try {

                        String[] messages = new String[100];

                        PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

                        BufferedReader in = new BufferedReader(

                           new InputStreamReader(socket.getInputStream()));

                        String inputLine, outputLine;

                        while ((inputLine = in.readLine()) != null) {

                                outputLine = "output>>" + SecretEncryptor.encrypt(inputLine
);

                                int length = outputLine.length();

                                out.println(outputLine);

                                if (outputLine.equals("Good Bye")) break;

                        }

                        out.close();

                        in.close();

                        socket.close();

                } catch (IOException e) {

                        log(e.getMessage());

                }

        }

        // Appends the given message into the log file
        private static void log(String mes) {

                try {

                        File logFile = new File("log.txt");

                        FileWriter out = new FileWriter("log.txt", false);

                        out.write(mes);

                } catch (IOException e) {

                }

        }

}
```

Appendix C – Seeded Defects

C1 - Client

1. Missing statement, line 11, main method is missing the keyword static
2. Extra statement, line 12, variable “validInput” not used
3. Wrong statement, line 26, Port number (4445 vs. 4444)
4. Missing statement, line 48, out stream should have been closed

C2 - Server (EncryptionServer)

5. Wrong statement, line 18, Boolean listening should have been true

C3 - Server (EncryptionServerThread)

6. Extra statement, line 15, array “messages” is never used
7. Extra statement, line 22, don’t need the variable length
8. Wrong statement, line 24, “Good bye” instead of “Bye”
9. Extra statement, do not need line 36
10. Wrong statement, line 37, for appending the constructor parameter should have been true, not false
11. Missing statement, line 38, nothing would be written to the file unless it is closed
12. Missing statement, line 39, no exception handling

Appendix D – Raw Data

D1 – Experiment Results – Real Groups

	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12
G1	1		1	1	1		1			1	1	
G2	1		1	1	1			1				1
G3			1	1	1	1		1		1		
G4	1	1	1	1	1		1	1			1	
G5	1		1	1	1		1	1		1		
G6	1		1	1	1							
G7	1		1	1	1			1			1	1
G8		1	1	1	1	1	1					
G9		1	1	1	1	1		1				
G10	1		1	1	1			1				

D2 – Experiment Results – Nominal Groups

D2.1 – Nominal Group #1

	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12
NG1	1		1	1	1	1	1	1		1		
NG2	1		1	1	1			1		1		
NG3		1	1	1		1	1	1				
NG4	1		1	1	1		1	1		1	1	
NG5		1	1	1	1	1		1			1	1
NG6			1	1	1	1	1	1	1		1	1
NG7		1	1	1	1	1		1		1	1	
NG8		1	1	1	1		1	1	1	1	1	
NG9		1	1	1	1		1					
NG10			1	1	1			1				

D2.2 – Nominal Group #2

	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12
NG1		1	1	1	1	1		1				
NG2			1	1	1		1	1	1	1		
NG3			1	1	1		1	1		1		
NG4		1	1	1	1		1	1	1	1	1	
NG5	1		1	1	1	1		1				1
NG6	1		1	1	1		1					
NG7		1	1	1	1	1	1	1				1
NG8		1	1	1	1	1		1		1	1	
NG9	1		1	1	1	1	1	1	1	1	1	
NG10			1	1	1			1				

D2.2 – Nominal Group #3

	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12
NG1			1	1	1			1				
NG2			1	1	1	1		1		1		1
NG3			1	1	1	1		1				
NG4			1	1	1	1	1	1	1	1	1	
NG5		1	1	1	1	1	1	1		1		1
NG6	1	1	1	1	1		1	1	1	1	1	
NG7	1		1	1	1	1	1	1		1		
NG8		1	1	1	1	1	1	1		1		
NG9		1	1	1	1	1		1			1	
NG10		1	1	1	1		1					

D2.2 – Nominal Group #4

	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12
NG1			1	1	1		1	1				1
NG2		1	1	1	1	1	1	1	1	1	1	
NG3			1	1	1		1	1		1		
NG4	1		1	1	1	1		1			1	
NG5			1	1	1	1	1	1	1		1	
NG6	1	1	1	1	1	1	1	1	1		1	
NG7	1	1	1	1	1	1	1	1				
NG8			1	1	1		1	1		1		
NG9		1	1	1	1	1		1		1		1
NG10			1	1	1			1				

D3 – Experiment Results – Individuals

	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12
I1	1		1	1		1	1	1				
I2		1	1		1	1						
I3			1									
I4			1	1	1		1	1		1	1	
I5				1	1							
I6				1	1			1				
I7			1	1	1		1					
I8			1	1	1	1	1	1	1		1	
I9			1	1	1							
I10			1	1	1			1				
I11		1	1	1		1	1	1				
I12				1	1			1		1		
I13			1		1					1		
I14			1		1		1					1
I15	1		1		1							
I16			1	1	1	1		1			1	
I17			1	1	1			1		1		
I18			1	1	1							
I19		1	1	1	1		1	1	1	1	1	
I20		1	1		1							
I21								1				
I22				1	1	1						1
I23			1	1	1			1				
I24					1			1				
I25	1		1		1							
I26			1	1	1	1		1				
I27		1	1	1				1			1	
I28			1		1		1	1		1		
I29												

Appendix E – Questionnaire and Code Explanation

E1 - Questionnaire

ID: ____

Question 1: Do you enjoy group work in general or do you enjoy working alone?

Question 2: Do you have any previous experience with software review?

Question 3: How would you rate your own programming skill level (scale 1-10)?

Question 4: Do you have any experience with this kind of programming (i.e. using sockets)?

Question 5: Did you feel that working in a group was beneficial for your performance or do you feel working alone would have been better?

Question 6: Were you confused about what was or wasn't a defect?

Question 7: If in a group: how did you agree on what was or wasn't a defect e.g. voting or discussion or something else?

Question 8: If working as an individual: do you think your performance would improve if put in a group setting?

Question 9: Do you have any other comments/thoughts/ideas not covered by these questions?

E2 – Code Explanation

Koden dere har fått utdelt inneholder tre klasser: EncryptionClient, EncryptionServer og EncryptionServerthread.

EncryptionClient inneholder programmet som er ansvarlig for å kommunisere med serveren. Når EncryptionClient kjøres antas det at EncryptionServer og EncryptionServerThread allerede er i gang. Det første klienten prøver å gjøre er å koble seg til serveren gitt et hostname og en port. Gitt at en forbindelse opprettes kan klienten nå kommunisere med serveren ved å gi serveren en streng som skal krypteres. Denne strengen blir sendt til EncryptionServer hvor den videre behandles i EncryptionServerThread. EncryptionServerThread krypterer strengen den mottar fra serveren og sender den tilbake kryptert.

Works Cited

Aurum, A., Petersson, H., & Wohlin, C. (2002). *State-of-the-Art: Software Inspections After 25 Years.* John Wiley & Sons, Ltd.

Bisant, D. B., & Lyle, J. (1989). *A Two Person Inspection Method to Improve Programming Productivity.* IEEE Transactions on Software Engineering.

Campion, M. A., Medsker, G. J., & Higgs, C. A. (1993). *Relations Between Work Group Characteristics and Effectiveness: Implications for Designing Effective Work Groups.* Personnel Psychology.

Fagan, M. E. (1976). *Design and Code Inspections to Reduce Errors in Program Development.* IBM Systems Journal, Vol 15, No. 3.

Fagan, M. E. (1986). *Advances in Software Inspection.* IEEE Transactions on Software Engineering.

Fairley, R. E. (1978). *Tutorial: Static Analysis and Dynamic Testing.* Fort Collins.

Johnson, P. M., & Tjahjono, D. (1998). Does Every Inspection Really Need A Meeting. *Empirical Software Engineering*, 9-35.

Knight, J. C., & Myers, E. A. (1993). *An Improved Inspection Technique.* Communications of the ACM, Vol 36, No. 11.

Kolawa, A., & Huizinga, D. (2007). *Automated Defect Prevention: Best Practices in Software Management.* Wiley-IEEE Computer Society Press.

Lanubile, F., & Visaggio, G. (1996). *Assessing Defect Detection Methods for Software Requirements Inspections Through External Replication.* Bari, Italy: University of Bari.

Marjara, A. S. (1997). *An Empirical Study of Inspection and Testing Data.* Trondheim: NTNU.

Martin, J., & Tsai, W. (1990). *N-Fold Inspection: A Requirements Analysis.* Communications of the ACM, Vol 33, No. 2.

McGibbon, T., Ferens, D., & Viennau, R. (2007). *A Business Case for Software Process Improvement Revised.* DACS.

Parnas, D. L., & Weiss, D. (1985). *Active Design Reviews: Principles and Practices.* London: Proceedings 8th International Conference on Software Engineering.

Porter, A. A., Votta Jr., L. G., & Basili, V. R. (1995). *Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment.* IEEE Transactions on Software Engineering.

Schneider, G. M., Martin, J., & Tsai, W. T. (1992). *An Experimental Study of Fault Detection in.* ACM Transactions on Software Engineering and Methodology.

Stålhane, T., & Husain Awan, T. (2005). *Improving the Software Inspection Process.* Trondheim: NTNU.

Votta Jr., L. G. (1993). *Does Every Inspection Need a Meeting?* ACM SIGSOFT Software Engineering Notes.

Wohlin, C., Aurum, A., Petersson, H., Shull, F., & Clolkowski, M. (2001). *Software Inspection Benchmarking - A Qualitative and Quantitative Comparative Opportunity.*

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2000). *Experimentation in Software Engineering - An Introduction.* Boston: Kluwer Academic Publishers.