

Confirmation Report:
Evaluating Verification and Validation Technologies
for Concurrent Components

Margaret Anna Wojcicki

Supervisor: Dr. Paul Strooper

14th February 2006

Abstract

Verification and validation (V&V) of concurrent components is a challenging task. There is no "silver bullet" for all V&V problems. It is therefore important to determine appropriate V&V solutions for particular contexts. Empirical evaluations can be a source of information regarding particular V&V technologies, but the studies are not useful on their own. Information from empirical studies needs to be presented in a format that classifies empirical and practical knowledge of V&V technologies and thus becomes a repository for practitioners. The purpose of this research is to devise a characterisation schema: a systematic approach to building up empirical and practical knowledge regarding the particular contexts and cost-effectiveness of V&V technologies for concurrent components. The schema itself will not be complete at the end of this research; it will have to continue to evolve based on practitioner needs and therefore it must be tailored for ease of use and ease of modification. Additionally, methodology needs to be devised for the continued development and evaluation of the schema.

1 Introduction

Verification checks whether a computer program conforms to its specification whilst validation checks if it meets the requirements/expectations of the client; together they are often referred to as V&V [80]. In this report, the term "V&V technologies" is used as a generic term referring to V&V techniques, tools and methods (this is a variation of the generic terminology

software engineering technology introduced by Birk [12]). V&V technologies may be dynamic (involving execution of the program) or static (not involving the execution of the program). This research focuses on the V&V of concurrent programs. A concurrent program specifies two or more processes (or threads) that cooperate in performing a task and each of those processes is a sequential program that executes a sequence of statements [3]. Processes cooperate through communication using shared variables or message passing. Concurrent programs are inherently more complex than sequential programs because they are non-deterministic and therefore do not always return the same output. Specific concurrency defects (section 2.1.1) further complicate the V&V of concurrent programs. These defects include interference, an interleaving of threads that results in incorrect updates to the state of a shared object, and deadlock, a situation in which there are no eligible actions to be performed by the program [63].

The focus of this research is the V&V of software components rather than software systems or programs. A software component, as defined by Szyperski [81], is a unit of composition with contractually specified interfaces and explicit context dependencies. It can be built from a set of atomic components. There are many V&V technologies specific to this context of concurrent components (section 2.1.2). They can be static or dynamic. Some focus on specific defects such as interference whilst others focus on deadlocks. Due to this diversity it is important to know when a V&V technology should be used.

Empirical research in software engineering has the promising ability to give practitioners the knowledge of contexts in which a particular V&V technology is most cost-effective. Empirical research on V&V technologies, despite its immaturity [11, 41], is one of the most active areas of empirical software engineering [79]. The first empirical studies, which date back about 25 years, are concerned with the comparison of V&V technologies. Comparison studies have since been replicated and performed in various contexts. Although these studies do not report that one particular V&V technology is clearly the most effective, they do continually recognise the complementary nature of the technologies and stress the importance of their combination. Most empirical studies of V&V technologies have thus far been applied to sequential programs, therefore at present there is a large gap in empiricism of V&V technologies in the specific context of concurrency. There are several technologies available for the V&V of concurrent components, but their empirical evaluation is sparse, therefore there is no evidence supporting when

they should be applied.

One such V&V approach is the TestCon method [60] which has been developed for the V&V of concurrent Java components by focusing on concurrency failures. It combines concurrency specific code inspection steps with automated static analysis tools [4, 37] and a dynamic analysis tool [59]. An overview of the TestCon method is found in section 2.1.3 of the Related Work. As part of the practical work completed for this research to date, the static analysis components of the method were evaluated through a comparison of code walk-through and automated static analysis tools with and without the application of the TestCon method steps. The study (summarised in section 3), although preliminary, did present some results that support the cost-effectiveness of combining automated static analysis tools with code inspection.

The TestCon method presents one approach to deal with V&V in concurrency, but it is essential to develop a repository of knowledge for practitioners that can be used to find cost-effective V&V solutions for their particular contexts. This repository of knowledge has to be accessible to both practitioners and researchers. It is most clearly depicted as a characterisation schema: a systematic approach to building up knowledge of V&V application through evaluation. A schema relies on an iterative approach of gaining information from empirical evaluation and practitioner experience and using this information to classify V&V technologies to the particular contexts in which they have been found most cost-effective. Such a schema is being developed for V&V technologies [83] and focuses on test case selection (one aspect of one V&V technology), but it needs to be expanded to deal with the particular issues of concurrent components.

The remainder of this report is broken down into three sections. Section 2 is an overview of literature in the fields of V&V for concurrent Java components and empirical evaluation of V&V technologies. Section 3 presents the practical work completed for this project to date. Section 4 reflects on the tasks and plan required to complete the research project.

2 Related Work

This review draws from the initial draft of the related work chapter of the thesis (available on request). The chapter contains a complete analysis of research in empirical software engineering (including guidelines, measure-

ments, body of knowledge and examination of empirical software engineering in practice) as well as V&V issues for concurrent components, current V&V technologies in software engineering and empirical evaluation of V&V technologies. The focus of the abbreviated review below is on the research needed in the development of a schema for the systematic use of V&V technologies for concurrent Java components. Therefore the review contains research in concurrency defects and the V&V technologies themselves to gain a perspective on the needs and challenges in this context. Secondly, research is pursued in the area of empirical software engineering to devise ways of evaluating the contents of the schema and determining when specific V&V technologies are most cost-effective in order to foster the schema's development.

2.1 V&V Issues for Concurrent Components

2.1.1 Defects

Defects of concurrent Java components are examined either as bugs (erroneous patterns of code) or failures (occurrence of incorrect program behaviour which is the result of a concurrency fault in code). Farchi et al. [27] document common concurrency bugs and classify them into three groups:

1. an erroneous assumption that a code segment is protected from access by multiple threads
2. an erroneous assumption that interleavings will not occur
3. blocking or dead thread bug pattern

This classification was used in the development of a benchmark consisting of mutant concurrent Java components. A benchmark allows researchers to compare any novel V&V technology to existing V&V technology, by running them on the same set of programs and then comparing their cost-effectiveness. It is important to keep in mind that a benchmark should be validated for completeness before it is used in empirical evaluations [47].

Concurrency failures have been classified for concurrent Java components through the use of a petri-net model [58]. Five transitions (specific to Java) were observed:

1. a thread requesting an object lock

2. a thread locking an object
3. a thread waiting on an object
4. a thread releasing an object lock
5. thread notification

If any of these 5 transitions fail to fire or fire erroneously they can lead to concurrency failures such as interference or deadlock. An examination of these transitions led to the development of the TestCon method that combines V&V technologies to cover all failures (see section 2.1.3). An understanding of concurrency bugs and failures can aid in the evaluation of V&V technologies when defects need to be seeded in components used in empirical studies.

Defects change as Java evolves. For example, in Java 5.0, some concurrency defects have been resolved (such as double-checked locking [5]) whilst novel defects may appear due to new ways of ensuring synchronisation as provided by the `java.util.concurrent` constructs [29, 53]. V&V technologies therefore have to evolve as well and it is then important to know if they are general or specific to a particular version of Java based upon the defects they are able to detect.

2.1.2 V&V Technologies

There are a number of diverse technologies available for the V&V of concurrent components, often specifically designed to deal with concurrency defects.

ConAn [59] is a tool that adapts Hansen's [31] approach of the systematic testing of monitors to perform dynamic analysis on concurrent Java components through deterministic execution. ConAn has recently been extended to handle interrupts, timed waits [84] and non-deterministic inputs and outputs [85]. There are other dynamic tools available that focus on deterministic testing (JaDA [9]) and checks for atomicity (Atomizer [28]). ConTest [23] is a tool for non-deterministic testing that can improve the detection of concurrency faults through instrumentation of the component under test. It uses heuristics to seed a component with delays causing primitives that can result in interleavings that can lead to interference.

Code inspection has been tailored to concurrency through the development of concurrent desk checking [33]. The process is designed to lower the

number of possible interleavings that are checked through the use of a selection process applied by a group of inspectors and can be made less tedious through the use of tools that support the review process and perform automated static analysis. Automated static analysis tools such as FindBugs [36, 37], Jlint [4] and E_Jlin [30] are run on bytecode to detect “bug” patterns [1] that can lead to concurrency defects. These tools are not guaranteed to detect (or report) all concurrency defects and they are prone to reporting false positives (erroneously reporting the existence of a defect).

TCGen [46] is a tool that automates test case generation for ADA with the testing criteria: edge coverage, loop coverage, interaction coverage as well as ordered sequence testing. Further work on test case generation is being done through the use of model checkers. As model checking becomes more automated, its use is becoming easier to adopt in industry. Formal specifications of components can be translated to model-checkers such as the Labelled Transition System Analyzer (LTSA) [63] that can subsequently check them. On the other hand, Java PathFinder [32] can translate Java programs to models for verification, but the process is still not completely trivial as it involves the use of assertions and the resulting error traces are in Promela as opposed to Java. Assertions can improve the testing process by checking if certain pre-conditions and post-conditions are true prior to method calls. In Java, this involves using the keyword `assert` followed by a boolean expression that is believed to be true when the assertion executes [38]. If the `assert` statement throws an error, the assumption believed of a method or statement is incorrect. Assertions have been successfully used in the testing of C [73], C++ [24] and Java programs [16], but it is not trivial to define appropriate boolean statements.

2.1.3 The TestCon Method: Combining V&V Approaches

As one can ascertain from the brief overview in section 2.1.2, V&V of concurrent components can be improved by combining methods. For example, ConTest is a tool that is run several times on a test suite combined with a defect detector such as a race-detection tool to report on potential problems. An evaluation of FindBugs, Jlint and E_Jlin [30] reported that these tools detect different classes of faults, suggesting that they are complementary and should be used together to cover a larger number of faults than when applied alone.

The TestCon method [56, 60], combines automated static analysis, code

inspection and dynamic analysis using the ConAn tool. It consists of the following nine steps:

Step 1: Execute FindBugs. Review reports of inconsistent synchronization bug patterns. Use step 2 to determine if any bug pattern detected by FindBugs is an actual error. FindBugs may also fail to identify a synchronisation fault.

Step 2: Inconsistent Synchronization. Ensure all shared variables are protected by synchronised blocks. Each access to a shared variable should be synchronised on the same object. Static shared variables should be synchronised on a static lock.

Step 3: Lock References. Ensure that lock references are not reassigned.

Step 4: Encapsulate shared variables. Shared variables that have package or public scope can be accessed outside the component; ensure that they are properly encapsulated to prevent outside access.

Step 5: Execute Jlint. Review Jlint reports of deadlock bug patterns. Use step 6 to determine if the bug pattern detected by Jlint is an actual error.

Step 6: Lock Graphs. For components with 2 or more locks, build a lock graph. Create a node for each distinct node used to synchronise a block. For each pair of nested synchronised blocks, draw a directed edge from the outer lock to the inner lock. A cycle confirms the possibility of a deadly embrace.

Step 7: Notifications. Browse the FindBugs report for the No (notify instead of notifyAll) bug pattern. Check for appropriate use of notify and notifyAll (notify wakes up at most one thread whereas notifyAll wakes up all threads).

Step 8: Condition Synchronisations. Browse the FindBugs report for bug patterns that involve calls to wait(). Examine the wait loop in a synchronised block. Determine if it has the desired synchronisation condition.

Step 9: ConAn. Use the ConAn tool to test the functional behaviour of the component and call completion times. To use the tool one must first identify test conditions, then construct test sequences, execute the test driver and finally analyse the tool's report.

Steps 1-4 mostly focus on defects that lead to interference, steps 5-6 focus on deadly embrace defects and the remaining steps focus on function requirements and correct process synchronisation.

In practice, most testing approaches involve some kind of methodology that combines V&V technologies; integration of different formats and sources of technologies is common to methods for various purposes [35]. The challenge is to develop an approach that employs the most cost-effective technolo-

gies for a particular set of circumstances. Such information can be gained through empirical and practical evaluation of the technologies in various contexts and determining the best combinations for those contexts.

2.2 Empirical Research in V&V

2.2.1 Comparison and Combination of V&V Technologies

The first comparison studies of V&V technologies date back to the 1970s with Hetzel [34] and Myers [66]. In the latter, Myers compared the effectiveness of code inspection versus testing and found that neither approach was significantly better. This led to the postulation of the Hetzel-Myers law [25]: “A combination of different V&V methods outperforms any single method alone”. It is important to note that there are exceptions to this postulation, despite the fact that Endres and Rombach refer to it as a law. The law is supported by the findings of the replicated studies of Basili and Selby [7], Kamsties and Lott [45], Wood et al. [87], and Juristo and Vegas [44]. These studies ran comparisons of structural testing, functional testing and code inspection; in some specific instances code inspection outperformed the other techniques, but the findings were not statistically significant. Interestingly each technique discovered different kinds of faults, thus providing more evidence for the benefits of complimentary V&V technologies. Empirical studies by Selby [76] and Wood et al. [87] evaluated the combination of V&V technologies and provided further support for the Hetzel-Myers law.

2.2.2 Empirical Research of V&V in Concurrency

Although empirical analysis is not new to V&V in software engineering, there are few experiments being conducted on V&V technologies used to detect concurrency failures. A case study was performed on a debugging tool for concurrent programs [68] and an empirical evaluation was conducted on deadlock detection technologies for Ada programs [20]. Promising results were observed in some recent case studies such as: (1) Constrained Specification-based (CSPE) testing for concurrent programs was empirically evaluated using faults produced by mutant generators [15], (2) a mutation-based case study was carried out on the TestCon method [57] to present its complete failure-detection capabilities, and (3) a case study was run on Atomizer, a dynamic checker for atomicity, on 12 benchmark programs [28].

None of these studies actually observed how the V&V technologies were applied by practitioners; (1) and (3) did not take into consideration how V&V technologies could be combined and the possible cost-effectiveness improvements due to the combinations. Thus far, combinations of V&V technologies for concurrent programs have been explored and advocated through analysis [88]. The empirical analysis of a combination of V&V technologies remains limited to evidence based on V&V technologies for sequential programs.

2.2.3 Challenges of Empirical Software Engineering

Software engineering is continuing to mature as a discipline and therefore in addition to theoretical research, the field has been expanding into empirical research. Since V&V technologies in software engineering are constantly evolving there must be ways to evaluate how cost-effective these technologies are. Empirical research has the ability to give confidence in existing and new approaches [11, 8]; unfortunately it is not often adopted. To date, empirical software engineering has been rather limited as only 1.9% of the published articles studied between 1993 and 2002 include controlled experiments [78] and many of these articles do not include sound statistical analysis or reflect high external validity [41]. The reasons for this lack of empirical research include the high costs of experimentation, the difficulty in applying the traditional scientific method in the software engineering context, the inability to extrapolate results into situations outside the experiment [69] and concerns that empirical research can slow the development of technology [40]. Another major difficulty is the connection between software engineering and its practitioners. In many ways, studies in this discipline depend on the social and psychological aspects of the users of V&V technologies and the variability of their skills and experience has an influence on experimental results [14, 40, 8, 72, 21]. Additionally, empiricism is often unpopular in software engineering, because researchers strive to discover and develop their own theories; they are not interested in confirming other people's results [69]. Also, they do not take advantage of empirical studies to direct research [70].

Many of the difficulties associated with empirical software engineering can be resolved through the use of guidelines as well as appropriate experimental design and it is well worth the effort. A software engineering process (such as a V&V technology) cannot be trusted to work effectively based solely on intuition [40, 8, 25, 70, 90]; the user of the V&V technology should have empirical evidence that the technology will work and therefore use it with

confidence and knowledge of its functionality [40, 6]. Experimentation is not a new concept to software engineers since V&V technologies have been empirically evaluated for 25 years [42] and experimentation has become an almost standard addition to research papers [70], but the empirical methodology needs to be improved [11, 13, 10]. Guidelines have been and are continually being developed in order to assist experimental design and data analysis; they influence aspects such as the measurements collected throughout the experiment [50] and the statistical analysis used to examine these measurements [51]. The desired outcome of this refined methodology is to be able to use empirical evaluation to (more confidently) extrapolate results into the real world (industry) through replication and meta-analysis [54, 48, 71] and to develop a body of knowledge for V&V technologies in software engineering [43]. Replicated studies, although unpopular due to their lack of the possibility of discovery, are the major source of evidence in science [54]. Single studies do not carry enough information to support or refute a theory. In fact replicated studies can expose errors in such studies that challenge the validity of the original results. Replication can be supported through the use of laboratory packages [62] and infrastructure [22] of experimental objects.

The greatest asset of scientific knowledge, that is, knowledge which has undergone experimentation to prove its factuality, is that it can also be predictive and applied in new situations [40]. Most importantly information gained from empirical evaluations and theoretical research needs to be useful to practitioners [75].

2.2.4 Applying Empirical Knowledge of V&V Technologies

As classification of faults and failures led to the development of V&V methods (such as TestCon [60], and heuristics for ConTest [27]) and repositories that evaluate them (such as the concurrent component benchmark [26]), so classification of the V&V technologies themselves can be used to determine their cost-effective application in practice. Vegas [82] proposed a test case selection strategy schema that classifies the strategies based on operational and tactical aspects. Tactical aspects include the goals of the test cases of a particular technique, whilst operational aspects of a technique are concerned with the conditions of its functionality. This characterization schema for test case selection strategies can be used to determine which strategy (or combination of strategies) is best suited for a particular situation (an approach advocated by Brinch Hansen [31]) more effectively than the use of

textbooks containing information on test case selection strategies [83]. Such information, as well as the theoretical analysis of V&V technologies and their combination [55] can help define a paradigm that can be used to customise V&V technologies for specific contexts to ensure the most cost-effective results. There is a gap in V&V technologies for concurrent Java components in the characterization schema, therefore part of this research project plan is to extend the schema to this context. Ideally, practitioners will be able to determine what technologies are most effective for their particular context by using validated empirical information encapsulated in contextual narratives [77].

3 Practical Work

This summarised account of a controlled experiment focusing on the TestCon method assumes familiarity with basic empirical terminology. The presentation follows the reporting guidelines for controlled experiments devised by Jedlitschka and Pfahl [39] (where applicable). Section 3.1 details the motivation, aims and goals of the study. Section 3.2 documents the related work. Section 3.3 presents the experimental design, hypotheses, data collection as well as the objects and subjects of the study. Section 3.4 documents the execution of the experiment. Section 3.5 combines the analysis and interpretation. Section 3.6 discusses the limitations of the study. Section 3.7 concludes this section and reflects on future work. A technical report (available on request) contains detailed information on the design and hypotheses, experimental entities, measures, power analysis, and experimental operation.

3.1 Motivation

3.1.1 Problem Statement

The TestCon method combines code inspection, automated static analysis and dynamic analysis in nine steps (section 2.1.3) [60]. The focus of the study is on the first eight steps of the method, which do not include dynamic analysis through the use of the ConAn tool. The use of the ConAn tool depends highly on the number and quality of test cases produced by its users [57] and therefore depends more on the skills and abilities of the users than the first eight steps and the automated static analysis tools [36, 4]. Since the TestCon method is a combination of V&V technologies, it is the

purpose of this study to observe whether the combinations of defect detection technologies for the V&V of concurrent components are more effective than using the technologies separately.

The reason for adding automation to the method is partially to improve its efficiency and the study can elicit if efficiency is indeed improved, but introducing automated static analysis tools also adds the issues of becoming familiar with the tool and learning to correctly analyse its output. Analysis using Jlint and FindBugs can result in false positives and users must be able to recognise the difference between an actual defect and a false positive. A user may also become too dependent on the automated static analysis tool and may not recognise defects that are missed by the tools.

The study also allows us to measure population variance which is needed to accurately estimate the sample size required to run replicated or similar experiments in the future with statistically significant results. Estimating the effect size can be done by dividing the observed mean difference expected by the observed standard deviation [19]. Then the sample size can be derived from a power analysis, significance level and the estimated effect size.

3.1.2 Research Objective

Following the template set out by the GQM [8] method, the research objective of this study is: Analyse ad-hoc and systematic V&V technologies as well as automated static analysis tools applied by novices for the purpose of comparison with respect to their cost-effectiveness from the point of view of the researcher and practitioner in the context of concurrent Java components.

3.1.3 Context

The experiment was run with participants enrolled in the Concurrent and Real-Time Systems course (3rd year course) at The University of Queensland. Each participant was given an individual training session on a particular V&V technique and/or tool and asked to detect defects in four concurrent Java components (detailed in section 3.3.4). Each participant was given a maximum of 60 minutes to apply the V&V method.

3.2 Related Work

3.2.1 Investigated Technology

The focus of evaluation in this study is the TestCon method (detailed in section 2.1.3 of the Related Work), as well as the automated static analysis tools Jlint [4] and FindBugs [37] that are part of the method.

3.2.2 Description of Alternative Solutions

There are many alternative V&V technologies for concurrent Java components and they are detailed in section 2.1.2 of the Related Work.

3.2.3 Related Experiments

Several empirical studies of V&V technologies have compared structural testing, functional testing and code inspection for sequential programs [7, 45, 65, 34, 87, 44]. This comparison study focuses on the major issues addressed by Selby [76] regarding the combination of V&V technologies, in a different context: concurrent programs and automated static analysis tools combined with code inspection. Automated static analysis tools have been previously evaluated as predictors of field failures [67].

Previous ventures into the empirical analysis of the TestCon method [2, 17] have focused on observing one subject's application of the method steps, but empirical evidence for TestCon's effectiveness and efficiency needs to show whether a population of subjects is going to use the method with consistent results. Additional empirical research in the context of concurrent components is documented in section 2.2.2 of the Related Work.

3.3 Experimental Design

3.3.1 Goals, Hypotheses, Parameters, and Variables

Applying the Goal Question Metric paradigm [7, 25, 40], the following goals of the study and their corresponding metrics have been defined:

1. Compare the effectiveness of the V&V technologies with respect to the percentage of seeded defects detected by each method.
2. Compare the effectiveness of the V&V technologies with respect to the percentage of false positives detected by each method.

3. Compare the efficiency of the V&V technologies with respect to their corresponding seeded defect detection rates.

The hypotheses of the study reflect the 2-Way ANOVA design (described in the following section). This design is reflected in the separation of the hypotheses into interaction and two factors based on the outline presented by Juristo and Moreno [40]. Each null hypothesis is denoted H_{0ij} , whilst each alternative hypothesis is denoted as H_{1ij} . The i corresponds to the goal identified and j is a counter in the case that more than one hypothesis is formulated per goal. This notation of hypotheses is recommended by the reporting guidelines for controlled experimentation in software engineering [39].

The template for each set of hypotheses and its corresponding metrics is defined as:

INTERACTION: Tool Use and TestCon

H_{0i0} : There is no interaction between tool use/no tool use and the method (TestCon/Control) in which it is applied.

H_{1i0} : There is an interaction between tool use/no tool use and the method (TestCon/Control) in which it is applied.

FACTOR 1: Automated static analysis tools vs. code inspection without tools

H_{0i1} : There is no difference in the <metric> of the static analysis tool and code inspection treatments.

H_{1i1} : The <metric> of the static analysis tool treatments is different from that of the code inspection treatments.

FACTOR 2: Control vs. TestCon application of tools/code inspection

H_{0i2} : There is no difference in the <metric> of the TestCon treatments and control treatments.

H_{1i2} : The <metric> of the TestCon treatments is different from that of the control treatments.

The metrics corresponding to the three goals of the study are indirect measures [50] defined as follows:

Percentage of seeded defects detected, PA:

$$PA = \frac{a}{s}$$

(a = number of seeded defects detected, s = total defects seeded)

Percentage of defects reported that were false positives, PF:

$$PF = \frac{f}{g}$$

(f = number of defects reported that were false positives, g = total number of defects reported)

Defect detection rate for seeded defects, RA:

$$RA = \frac{a}{t}$$

(a = number of seeded defects detected, t = time required to apply the V&V technology)

The goal of determining the effectiveness of a V&V technology is achieved through the measure of seeded defects detected. A V&V technology is meant to detect defects so that they can be resolved prior to the component being used. The more defects a technology is likely to detect, the more effective it is. Effectiveness of a V&V technology also depends on the number of false positives it detects. A V&V technology is more effective when it reports fewer false positives so that time is not spent on resolving issues that are not defects. Similarly, a V&V technology is more efficient than another if it can detect defects in less time. Efficiency is related to the cost of applying a V&V technology; an inefficient technology is costly to apply.

Motivation and self-perceived mastery of the V&V technologies were also recorded throughout the experiment. Motivation was measured by the subject's response on a data collection form based on an ordinal scale from 0 to 5. The motivation scale and question is the same as that in the experimental package presented by Lott [62, 61]. Mastery of the treatment V&V technology was measured through the subject's response on a data collection form based on an ordinal scale and question from Lott's experimental package. Kamsties and Lott [45] examined these characteristics in order to see if better results in the application of certain V&V technologies were due to the motivation for taking part in the study or the mastery of the V&V technologies of the participants, as opposed to the benefits of applying the technology. The following hypotheses correspond to the goals related to the motivation and mastery of the technology:

4. Determine if detection effectiveness depends on the motivation of the participant to take part in the study.

H_{04} : The participant's motivation has no correlation with their performance (in terms of the percentage of seeded defects detected).

H_{14} : The participant's motivation is correlated with their performance.

5. Determine if the detection effectiveness depends on the participant's self-perceived mastery of the treatment.

H_{05} : The participant's mastery of the V&V technology has no correlation with their performance.

Table 1: Summary of Experimental Design.

Technology/Type	Control	TestCon
Code Inspection	4 subjects	4 subjects
Tool Use	4 subjects	4 subjects

H_{15} : The participant’s mastery of the V&V technology is correlated with their performance.

3.3.2 Experiment Design

As depicted in Table 1, each individual taking part in the study applied the tools and the particular code inspection technique through the following treatments:

1. code inspection without the steps of TestCon
2. code inspection with the steps of TestCon
3. tool use without the steps of TestCon
4. tool use with the code inspection steps of TestCon

The treatments with the steps of TestCon refer to defect detection applied following a number of defined steps either solely related to the use of automated static analysis tools or code inspection. The treatments without the steps of TestCon are ad-hoc.

Therefore the main effects due to the V&V task (tool use vs. code inspection) and the main effects due to the V&V method (control vs. TestCon) can be assessed. Note that treatment 4 only has the steps of TestCon that are associated with the automated static analysis tools so that the tool use can be compared without the additional steps of code inspection. It can therefore be observed whether using the tools with code inspection steps is more effective than using the tools alone without additional steps. The TestCon steps (see Section 2.1.3) shared by treatments 2 and 4 are step 2 (in treatment 4 this step included both steps 1 and 2 of the TestCon method), step 6 (in treatment 4 this step included both steps 5 and 6 of the TestCon method), and steps 7 and 8. The order of application of the steps is also different for these steps in treatments 2 and 4 (treatment 2 order: 2, 6, 7, 8; treatment 4

order: 2, 7, 8, 6) so that the steps are ordered based on using FindBugs first and then Jlint.

A power analysis was conducted to determine the appropriate sample size for the experiment. Based on Cohen's [19] sample size analysis, hypothesis testing of ANOVA with an F test ($\alpha = .10$, Power = .80) for an effect size of .80 requires a minimum sample size of 4 per group (for a study with 4 groups this means a total of 16 subjects). Cohen's effect size values for tests on means in the analysis of variance are .10 (small), .25 (medium) and .40 (large). The effect size was determined using estimated mean and variance values; it is rather large but it will therefore provide some practical significance.

3.3.3 Subjects

The experiment was planned to be run with sixteen students enrolled in the Concurrent and Real-Time Systems course (3rd year course) at The University of Queensland. The students volunteered to take part in the study and they were paid to take part in the study for 90 minutes. Since the students volunteer to participate in the study the sample of the population is defined as a convenience (rather than a random) sample from the population of novice software engineers.

3.3.4 Objects

The V&V technologies were applied to four concurrent Java components (`Mutex`, `FIFOReadWriteLock`, `ReadersWritersSync`, `Piper`). The first two components are modified versions of the `java.util.concurrent` package (release 1.3.4) [52] components, the third is a component specifically designed for the study using the `FIFOReadWriteLock` component from the same package, and the last is a component from the concurrent Java components benchmark [26]. Defects were seeded to make sure that all steps of the `TestCon` method could be analysed. Eight defects in total were seeded among the components and the first and fourth components had one defect each prior to seeding. Six of the ten defects were reported by the automated static analysis tools whilst four were only detectable by inspection of the code.

3.3.5 Data Collection Procedure

The experiment was run during the second half of the course once all participants would have been introduced to the concurrency concepts needed

to complete the study. All data was collected on sheets modified from the existing templates available from the lab package [61] of the Kamsties and Lott experiment [45]. This included an information sheet to be completed prior to applying the V&V technology to document the experience of the participants. Then each participant filled in a defect detection report to list the defects detected, the time at which they were detected and the duration of the step of the TestCon method (if applicable). Participants that applied the FindBugs and Jlint tools listed all the concurrent defects that the tools reported and noted whether or not they were false positives. Additionally information was collected by the experimenter regarding the time spent applying the technology (in full) and the time spent applying the Jlint and FindBugs tools. Following the experiment, the participants were asked to report their perceived mastery of the V&V technology as well as whether or not they conformed to the particular process presented in training. Data was also collected on how the participants applied the tools (whether or not they used special settings or filters).

3.4 Execution

3.4.1 Sample

Sixteen students enrolled in the Concurrent and Real-Time Systems course at the University of Queensland participated in the study. All participants were introduced to concurrency aspects in Java so that they should understand the defects in the experiment. On the information sheet, they reported an average of 3 years experience in Java (minimum 0.2, maximum 5) and listed an average of 4 courses in Java (including Concurrent and Real-Time Systems) with a minimum of 1 and a maximum of 8 courses. No participants that used the automated static analysis tools reported prior experience in the use of Jlint and FindBugs or any other static analysis tools. None of the participants reported using special settings or filters when applying the tools. Few of the participants performing a code inspection treatment had taken a course including material on code inspection (minimum 0, maximum 2). Confidentiality of the information sheets was assured by filling in a numeric subject identifier on the set of information sheets for each participant.

3.4.2 Preparation

A set of cards representing each of the four participants for each of the four treatment groups was used for the randomization of the participants to the groups. Prior to each run of the study one of these cards was randomly selected to determine the treatment the participant would apply (except close to the end of the experiment when fewer cards were available).

Participants received training in the particular V&V technology that they applied. This 20 minute training procedure consisted of a scripted presentation including slides presented to the individual outlining the V&V technology (each participant also had a copy of the slides to peruse throughout the study). Secondly, each participant was introduced to the four components by an overview of their corresponding interfaces (again using slides available to the participant). Lastly, participants were familiarised with the components by perusing them without applying a V&V technology to ensure that they understood all the programming constructs.

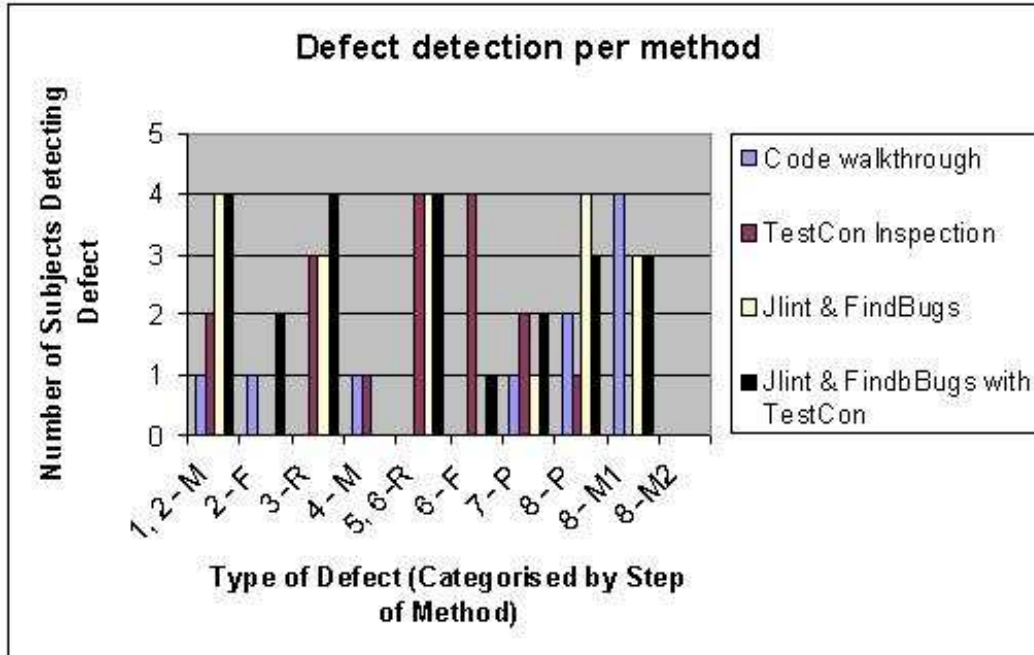
3.5 Analysis and Interpretation

Statistical analysis was performed using the SPSS statistical package (Version 11.5). Despite the fact that the sample is a convenience rather than a random sample and the sample size is low, a parametric ANOVA analysis was run given that the results had a normal distribution and there was homogeneity among treatment groups according to the SPSS analysis on these factors [18]. Additionally, ANOVA analysis has been shown as robust, therefore it is relatively insensitive to violations of the assumption of normality as well as the assumption of equal variances [64]. The correlation between motivation and effectiveness of testing technique was done using the Spearman rank-correlation test as it dealt with ordinal values.

3.5.1 Analysis of Effectiveness

The data regarding the number of participants detecting a defect per treatment group is presented on the chart in Figure 1. On the whole treatments 2, 3 and 4 outperform treatment 1 and no treatment detected defect 8-M2 (related to the condition of the loop containing a wait statement). From the chart in Figure 2, the combination of TestCon with tools does as well or outperforms TestCon alone for all steps except for step 6 (a deadly embrace

Figure 1: Defects detected per treatment. Defects are classified according to the numbers of the steps of the TestCon method followed by the first letter of the component they are in (M = Mutex, F = FIFOReadWriteLock, R = ReadersWritersSync, P = Piper) and the number of the defect (if there are multiple defects of the same type in the component).



defect that can only be detected through code inspection). The result for this step 6 may be due to a dependence on the automated static analysis tool that prevents the user from applying code inspection thoroughly.

Table 2 presents the results of a Two-Way ANOVA of the defect detection effectiveness. The bottom row in Table 2 shows that we cannot reject H_{010} since the F value is less than 1 and therefore there the study does not show interaction between Tool Use and the application of TestCon ($p = 0.688; p > 0.10$). The F value in the TESTCON row is greater than 1, suggesting that there is more variation between groups than within groups, from which one can infer that there is a difference in the effectiveness of TestCon steps applied with tools or code inspection in terms of the percentage of seeded defects detected. However H_{012} is not rejected ($p = 0.157; p > 0.10$) as the result is not statistically significant. The F value related to the appli-

Figure 2: Defects detected by steps in treatments 2 and 4. Defects are classified according to the numbers of the steps of the TestCon method followed by the first letter of the component they are in (M = **M**utex, F = **F**IFOReadWriteLock, R = **R**eadersWritersSync, P = **P**iper). The two defects that are not depicted in this chart (3-R and 4-M) are left out because they do not correspond to steps shared by both treatments.

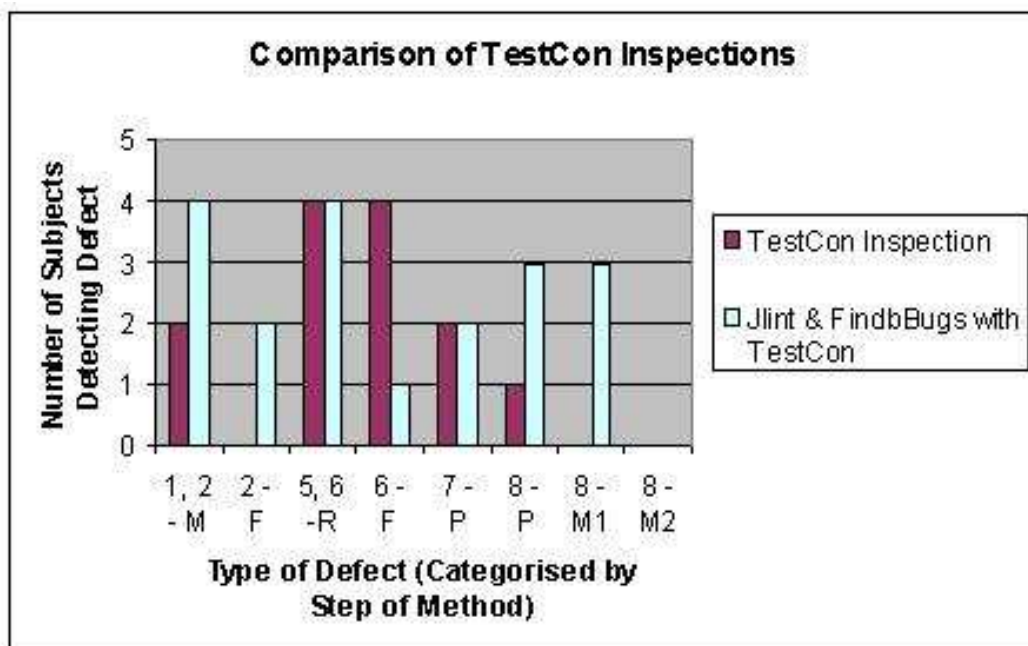


Table 2: Two-way ANOVA of the defect detection effectiveness with respect to the percentage of seeded defects detected.

Source	Type III SS	df	Mean Square	F	Sig.
TOOLUSED	.106	1	.106	3.189	.099
TESTCON	.076	1	.076	2.283	.157
TOOLUSED * TESTCON	.006	1	.006	.170	.688

Table 3: Defect detection effectiveness summary of descriptive statistics - mean (standard deviation) in which all values correspond to percentage of seeded defects detected. The Columns row contains the averages for all treatments not using tools (TOOLUSED -N) and those using tools (TOOLUSED - Y). The Rows column contains the averages for all treatments using TestCon steps (TESTCON - Y) and those not using the TestCon steps (TESTCON - N).

	TOOLUSED - N	TOOLUSED - Y	Rows
TESTCON - N	.250 (.1732)	.450 (.1000)	.350 (.1690)
TESTCON- Y	.425 (.1500)	.550 (.2646)	.488 (.2100)
Columns	.338 (.1768)	.500 (.1927)	.419 (.1974)

cation of automated static analysis tools is greater than 1 and H_{011} is rejected as shown in the TOOLUSED row of the Table 2 ($p = 0.099; p < 0.10$). Table 3 shows the descriptive statistics related to the Two-Way ANOVA of defect detection effectiveness based on percentage of seeded defects detected. In that table one can see that using automated static analysis tools is more effective than just doing code inspection or code walk-through as the percentage of seeded defects detected is greater than when tools are not used since the values in column TOOLUSED-Y are consistently greater than those in column TOOLUSED-N. The values of rows TESTCON-Y are greater than TESTCON-N showing that the defect detection technique that applies TestCon steps detects a higher percentage of the seeded defects, but this result is not statistically significant since H_{012} was not rejected.

Table 4 presents the results of an ANOVA analysis on the effectiveness in terms of the percentage of defects reported that were false positives. Although the F value in the bottom row of Table 4 for the interaction of tool use and TestCon variables is slightly greater than 1, H_{020} is not rejected ($p = 0.209; p > 0.10$), therefore the study does not show interaction in terms

Table 4: Two-way ANOVA of the defect detection effectiveness with respect to the percentage of false positives detected.

Source	Type III SS	df	Mean Square	F	Sig.
TOOLUSED	.322	1	.322	11.900	.005
TESTCON	.122	1	.122	4.519	.055
TOOLUSED*TESTCON	.048	1	.048	1.762	.209

Table 5: Defect detection effectiveness summary of descriptive statistics - mean (standard deviation) in which all values correspond to percentage of false positives detected. The Columns row contains the averages for all treatments not using tools (TOOLUSED -N) and those using tools (TOOLUSED - Y). The Rows column contains the averages for all treatments using TestCon steps (TESTCON - Y) and those not using the TestCon steps (TESTCON - N).

	TOOLUSED - N	TOOLUSED - Y	Rows
TESTCON - N	.7056 (.12620)	.3215 (.14232)	.5090 (.24423)
TESTCON- Y	.4214 (.22820)	.2468 (.13605)	.3341 (.19906)
Columns	.5635 (.22849)	.2797 (.24423)	.4216 (.23342)

of the percentage of false positives detected. The F values in the TOOLUSED and TESTCON rows are both greater than 1 and therefore interestingly, both H_{021} ($p = 0.005; pp < 0.10$) and H_{022} ($p = 0.055; p < 0.10$) are rejected as these values are statistically significant. Table 5 shows that when automated static analysis tools and the TestCon method steps are applied the percentage of false positives reported is lower; The TOOLUSED-Y and TESTCON-Y rows are consistently lower than TOOLUSED-N and TESTCON-N.

The number of false positives correctly and incorrectly identified by the testers using automated static analysis tools is presented in Table 6. In the treatments that involved the application of automated static analysis tools, testers were asked to report all the defects detected by the tools and to indicate whether or not they believed the defects were false positives. Jlint and FindBugs used with TestCon slightly outperform the tools used on their own because fewer false positives are identified incorrectly. On average, about 6 false positives were correctly identified by Jlint and FindBugs users compared to 4.5 false positives for Jlint and FindBugs users with TestCon. This value indicates that the testers without TestCon may end up focusing

Table 6: Descriptive statistics of false positives incorrectly identified for treatments using automated static analysis tools. Y means an actual defect erroneously identified as a false positive. N is a defect identified as an actual defect that is a false positive.

False Positives/Treatments	Jlint and Find-Bugs Mean (Std Dev)	Jlint and Find-Bugs with TestCon Mean (Std Dev)
FP Incorrectly Identified (N)	2.25 (1.89)	1.25 (0.5)
FP Incorrectly Identified (Y)	1.5 (0.58)	1.25 (1.5)

on defects that are false positives, thus preventing them from focusing on actual defects.

3.5.2 Analysis of efficiency

Table 7 shows there is no significant interaction between tool use and TestCon application in the bottom row (F value is greater than 1 but $p > 0.10$), therefore H_{030} is not rejected. Although the F value for the TOOLUSED row is much greater than 1, the result is not quite significant ($p = 0.106$; $p > 0.10$) and H_{031} is not rejected suggesting that defect-detection rate is not affected by the use of automated static analysis tools. Table 8 does show that when tools (the TOOLUSED - Y vs the TOOLUSED - N columns) were used the defect-detection rate (in terms of seeded defects detected per hour) was higher (with high variability). The F value is lower than 1 in the TESTCON row of Table 7 and H_{032} is not rejected, therefore defect-detection rate is not shown to be affected by the application of TestCon steps. The total defect-detection times of the treatments show that applying TestCon usually takes more time than applying automated static analysis tools alone or ad-hoc code walk-through (see Table 10). The application times of the steps of the methods show that steps 7 and 8 (that are concerned with the appropriate use of notification and condition synchronisations) take less time on average when applying code inspection with the automated static analysis tools (see Table 9). Step 6 (the deadlock step with lock graphs) also takes slightly less time with automated static analysis tools than code inspection alone.

Gross defect detection rate refers to the defect-detection rate of a treatment that includes the detection of both seeded defects and false positives.

Table 7: Two-way ANOVA of the defect detection efficiency.

Source	Type III SS	df	Mean Square	F	Sig.
TOOLUSED	18.657	1	18.657	30.53	.106
TESTCON	.393	1	.393	.064	.804
TOOLUSED*TESTCON	7.483	1	7.483	1.225	.290

Table 8: Defect detection efficiency. A summary of descriptive statistics - mean (standard deviation). All values correspond to the seeded defect detection rate (seeded defects detected per hour). The Columns row contains the averages for all treatments not using tools (TOOLUSED -N) and those using tools (TOOLUSED - Y). The Rows column contains the averages for all treatments using TestCon steps (TESTCON - Y) and those not using the TestCon steps (TESTCON - N).

	TOOLUSED - N	TOOLUSED - Y	Rows
TESTCON - N	3.6223 (2.93185)	7.1497 (2.65839)	5.3860 (3.20432)
TESTCON- Y	5.3034 (1.89792)	6.0953 (2.27549)	5.6994 (1.98545)
Columns	4.4628 (2.45666)	6.6225 (2.35912)	5.5427 (2.58019)

The average gross defect detection rates show that using TestCon in code inspection greatly lowers the defect detection rate, but that rate is increased to match the rate of ad-hoc code walk-through when combined with static analysis tools (see Table 10). The static analysis tools applied alone have the greatest defect-detection rate, but the many defects detected may actually be false-positives and there is little (if any) inspection of the code to determine if the tools missed defects. Note that in a lot of these descriptive statistics the standard deviation is quite high, therefore further studies with large sample sizes would need to be done to confirm these results and allow for tests of statistical significance.

3.5.3 Analysis of motivation and mastery of V&V technology

Correlation values indicate that performance cannot be predicted by the self-reported motivation and mastery of V&V technology values. The R value for the motivation is 0.344 and this value is lower than the critical r at the significance criterion 0.05 [89]. Since the mastery of defect-detection technology was asked separately for the use of Jlint and FindBugs there are

Table 9: Descriptive statistics of the time (in minutes) of steps in the TestCon treatments. (Treatment 2/Treatment 4)

Steps/ Treatments	Step 2 Mean Time (Std. Dev.)	Step 6 Mean Time (Std. Dev.)	Step 7 Mean Time (Std. Dev.)	Step 8 Mean Time (Std. Dev.)
TestCon Code Inspection	3.19 (2.12)	3.37 (2.50)	4.83 (5.35)	4 (4.36)
Jlint and Find- Bugs with Test- Con	3.35 (1.68)	2.17 (0.99)	2.69 (1.46)	2.06 (1.01)

Table 10: Descriptive statistics of the total defect detection time and gross defect detection rate.

Treatment	Average Time (hour) [Std Dev]	Average Gross Defect De- tection Rate (per hour) [Std Dev]
Code Walk-through	.76 [0.26]	13.08 [8.39]
TestCon Code Inspection	.81 [0.12]	8.52 [1.81]
Jlint and FindBugs	.66 [0.13]	21.78 [4.29]
Jlint and FindBugs with TestCon	.88 [0.17]	13.69 [2.55]

two R values: 0.059 and 0.111, both these values are also lower than the critical r value at significance criterion 0.05. H_{04} and H_{05} are therefore not rejected.

3.6 Limitations of the Study

Validity threats of the study are discussed in the following sections categorised as internal, external, conclusion and construct validity [86].

3.6.1 Internal validity

Since the study was conducted on volunteer participants, as opposed to a random sample of students (or practitioners), the reliability of the performance results in the study may depend on their willingness to take part in the study. Someone enthusiastic and willing to spend approximately 90 minutes for the study may perform more effectively than an individual unwilling to give up their time for the same purpose, although the motivation analysis results show that motivation did not influence performance. The experimental environment was a meeting room with only the participant and experimenter present, which may affect the focus of the individual as it may make them feel as if it is an examination-like situation and this can influence their performance. Although each participant was trained individually, care was taken to script and provide slides to the participants so the training they received would be as consistent as possible.

The participant number can also greatly influence the results of the study. The fewer participants taking part in the study, the more likely that highly skilled individuals will be grouped in a particular treatment that can depend more on their skills than the actual V&V technology. Although the group of individuals taking part in this study were as homogeneous as possible (they were all from the same course and should therefore have the same amount of experience regarding concurrency defects) their skill variability could still influence the results.

3.6.2 External validity

The study was not conducted on a random sample of software engineers, greatly reducing its external validity. It was also not conducted on practitioners of the V&V technology, although the results can be generalised to

software engineering novices [51] who have just been introduced to Java concurrency.

The concurrent Java components may not be representative of those found in industry as they are relatively small (60-105 LOC) and one of them was a modified student program, whilst two others were the result of splitting up another more complex component. For the purpose of examining the V&V technologies at the unit (rather than integration or system) level, these components should be sufficient. Two of them were modified components of the `java.util.concurrent` package [52].

The total number of lines of code inspected was approximately 137 (not including comments and lines with 1 or less characters, i.e., braces) and subjects were given 60 minutes to inspect the code which is consistent with Russel's [74] findings which recommend that no more than 150 lines of code be inspected per hour. The limit of 60 minutes may not reflect the time that software engineering professionals would spend on defect detection as they may not adhere to strict limits. Additionally, this time limit may influence the results as some subjects may feel they need to complete the task in the 60 minutes provided; this can influence their speed of defect detection as they may be trying to complete the task in the given time. Some subjects may believe that they have to use up all the allotted time and this may not reflect the actual defect detection rate of the treatment applied.

The seeded defects may not be representative of defects found in industry, but care was taken to make them representative of concurrency defects and to ensure that the 8 relevant steps of the TestCon method could be evaluated.

3.6.3 Conclusion validity

Effectiveness was measured by the percentage of actual defects found and the percentage of defects reported that were false positives. Efficiency used the additional measure of time of the application of the V&V technology. These measures can be repeated and lead to high conclusion validity. On the other hand, the participant's motivation and mastery of V&V technology scores are subjective and may give very different responses in other situations. Mastery of the V&V technology can be determined more effectively through an objective test but time constraints made it difficult to collect such data. As in the Kamsties and Lott study [45], data on the mastery of the V&V technology was collected after it was applied, thus ensuring the participants had a better idea of their ability at applying the technology than they would

after the training alone.

3.6.4 Construct validity

Effectiveness and efficiency are common measures applied in the evaluation of V&V technologies. Motivation was previously used in the Kamsties and Lott [45] study to examine the effects of motivation on performance. Mastery of the V&V technology was a measure recommended by the same people.

3.7 Conclusions

3.7.1 Discussion of Results

The results of the experiment show that the use of static analysis tools (such as Jlint and FindBugs) can be more effective than code inspection (with or without the TestCon steps). This may be due to the fact that the participants are not familiar with the defects that need to be detected and therefore cannot recognise them in the code. Experienced practitioners should not have this problem, but even so, they may forget certain defects or fail to recognise them, therefore the tools can still prove to be useful.

Based on the power analysis, the number of participants in the study needed to be 16 for an estimated effect size (Cohen's f [19]) of 0.8 or higher. However the effect size observed in the study, applying Cohen's effect size equation [19] to the sample means and variance (of percentage of defects detected), was actually 0.5 which requires 7-10 participants per treatment group (28-40 in total). An analysis of the descriptive statistics associated with the defects detected per V&V technology shows that combining TestCon with automated static analysis tools improves effectiveness, but this data needs to be supported by a future study with statistical analysis and a larger sample size.

In terms of the percentage of defects reported that were false positives, both the use of static analysis tools and the TestCon method proved more effective. The application of the TestCon method and the tools makes participants focus on actual defects. The observed effect size was 0.8 which suggests that a treatment group size of 4 is sufficient. Of course replicated studies would still benefit from a higher participant number.

Efficiency is not improved for the participants that applied static analysis tools and the TestCon method; this may actually be due to the size of the

components analysed (especially with respect to the tool use). Defect detection on larger components may show that the use of static analysis tools is more efficient. An analysis of the descriptive statistics shows that applying TestCon with automated static analysis tools increases the defect-detection rate when focusing on individual TestCon steps and gross defect-detection rate.

3.7.2 Lessons Learned

The most significant lesson learned from the design of the experiment is to use power analysis when being limited in sample size. Additionally being aware of the existing body of knowledge (such as lab packages and experiences) in the context of V&V technologies helps a researcher be aware of the gap they are filling. Although the study run was relatively small, it can contribute to the full body of knowledge and it does not have to generalise to all possible contexts. Instead it can be a stepping-stone combined with the data collected in other small studies. It is also important to maximize the information available from the study (as advocated by Juristo and Moreno [40]). Designing the study as a 2-Way ANOVA, for example, gave more insight into the two different components of the TestCon method (tool use and systematic code inspection) and facilitated the use of smaller groups for the treatments.

Lessons were also learned regarding data collection. Listing courses was not a very accurate measure of skills in Java as many students could not remember the courses they took or they took them at different institutions thus making it difficult to add up courses. Lastly, manual collection of data, such as time, makes the statistical analysis of the study time-consuming and suffers from possible bias based on the interpretation of the collection forms. Recording the application time of the TestCon method steps automatically can save participant application time and therefore be a more accurate reflection of the costs involved.

3.7.3 Relation to Existing Evidence

Although not all the results regarding cost-effectiveness were statistically significant, overall there is a correspondence with the findings of this study and that of Selby [76] and Wood et al. [87] regarding combinations of V&V technologies. This study did not apply post-hoc comparison of combinations of

V&V technologies, instead it applied a 2-Way ANOVA design to perform the comparisons. Importantly, the study provided examination of V&V technologies in the context of concurrent Java components with a number of subjects (as opposed to one subject in [2]) taking into account their variation in skills.

3.7.4 Future Work

Further research in this context can be done through empirical studies examining different combinations of V&V technologies (such as dynamic analysis) for concurrent components. Additionally a more realistic context in terms of participants (and greater participant numbers) and environment, components and defects (specifically real vs seeded defects) can improve the external validity of a future study. It is also vital to be more aware of what is actually advocated in the practice of V&V application in this context, so a useful comparison can be made, specifically of novel vs currently used V&V technologies.

The ease of replication of this study can be greatly increased through the development of reusable experimental constructs such as automated data collection and marking. Skills can be measured more accurately through the use of objective tests. Objective testing and automated data collection can also reduce the experimental bias [49].

4 Plan

The aim of this research is to contribute to the V&V of concurrent Java components and empirical software engineering in three ways:

1. Devise a characterisation schema; a systematic approach to building up empirical and practical knowledge. The knowledge contained in this schema concerns the particular contexts and cost-effectiveness of V&V technologies of concurrent components. A characterisation schema organises information by classifying it in a format that helps practitioners decide when a given V&V technology is useful. The current schema [83] for testing technology focuses on test case selection. Since not all V&V technologies of concurrent components involve test case selection, it is a challenge to expand the schema to include technology such as code inspection. The schema can then:

- make use of the empirical knowledge in practical V&V technologies for concurrent components
 - inform researchers of gaps in empirical research regarding V&V technologies for concurrent components
 - inform researchers of the empirical methodology recommended in this context (and provide support for the empirical research with reusable constructs)
2. Determine the needs of practitioners that apply V&V for concurrent Java components. This information can guide future empirical evaluations and the development of the characterisation schema. Additionally it can guide the modification of the TestCon method itself.
 3. Modify the TestCon method for the V&V of concurrent Java components; generalise it for use in both previous and current versions of Java. The characterisation schema will further expand the use of the method thus resulting in a more modular method ensuring that cost-effective combinations of the V&V technologies of the method can be derived for specific contexts.

It is important to evaluate the characterisation schema and this will be done in a two-fold approach of controlled experimentation and case study. At this time it is unclear what kind of study will be performed first, as both forms of evaluation are recommended. According to Juristo, Moreno and Vegas [42] a case study in an industrial context takes place after controlled experimentation once the laboratory environment of evaluation has shown some results and provided a dry run for the study. Endres and Rombach [25] on the other hand advocate the case study approach first to become aware of what vital questions need to be asked in the laboratory study or controlled experiment. Currently the plan incorporates the former approach of evaluation.

The plan is outlined in the Gantt chart (Figure 3) with descriptions of the necessary tasks and their estimated duration in Table 11. All tasks include the write up necessary to document the work done and the associated results collected (if any). Contribution 1 is related to tasks 4-7 as it includes the development of the characterisation schema, its evaluation (using reusable experimental constructs) and its modification. Contribution 2 is related to

Table 11: Research Project Tasks and Estimated Duration

Task Number	Approx. Weeks to Complete	Description of Task
1	8	Devise a practitioner survey to determine requirements of the schema - release and collect results for 4 weeks
2	4	Analyse results from Preliminary Study and how they should affect the TestCon method and how it should influence future empirical studies in this context; write up and submit to ISESE'06 Conference
3	10	Modify TestCon method (generalise to Java 5 and make additional changes based on empirical information from the first experiment)
4	10	Design and implement reusable experimental constructs
5	20	Expand the current characterisation schema for testing techniques using practitioner survey information and incorporate V&V technologies of the TestCon method into it
6	30	Outline the empiricism needed in order to develop the schema further and evaluate it using controlled experiments and case studies; Plan and run controlled experiment to evaluate the new approach
7	36	Modify schema (and possibly empirical methods) based on experimental results; plan and run case study to evaluate it
8	16	Complete thesis
9	104	Update related work based on newly released (or discovered) literature

- [6] V. R. Basili. The role of experimentation in software engineering: Past, current, and future. In *Proceedings of the 18th International Conference on Software Engineering (ICSE)*, pages 442–449, 1996.
- [7] V.R. Basili and R.W. Selby. Comparing the effectiveness of software testing strategies. *IEEE Transactions in Software Engineering*, 13(12):1278–1296, 1987.
- [8] V.R. Basili, R.W. Selby, and D.H. Hutchens. Experimentation in software engineering. *IEEE Transactions in Software Engineering*, 12(7):733–743, 1986.
- [9] A. Bechini and K-C. Tai. Design of a toolset for dynamic analysis of concurrent Java programs. In *Proceedings of the 6th International Workshop on Program Comprehension*, pages 190–197, 1998.
- [10] A. Bertolino. Software testing research and practice. In *Proceedings of the 10th Int. Workshop on Abstract State Machines (ASM 2003)*, pages 1–21, 2003.
- [11] A. Bertolino. The (im)maturity level of software testing. In *Proceedings of the Workshop on Empirical Research in Software Testing (WERST 2004)*, pages 1–4, 2004.
- [12] A. Birk. Modelling the application domains of software engineering technologies. Technical Report IESE 014.97/E, Fraunhofer IESE, Kaiserlautern, Germany, 1997.
- [13] L. Briand and Y. Labiche. Empirical studies of software testing techniques: Challenges, practical strategies, and future research. In *Proceedings of the Workshop on Empirical Research in Software Testing (WERST 2004)*, pages 1–3, 2004.
- [14] J. Carver, J.V. Voorhis, and V. Basili. Understanding the impact of assumptions on experimental validity. In *Proceedings of the 2004 International Symposium on Empirical Software Engineering*, pages 251–260, 2004.
- [15] R.H. Carver and K-C. Tai. Use of sequencing constraints for specification-based testing of concurrent programs. *IEEE Transactions in Software Engineering*, 24(6):471–490, 1998.

- [16] Y. Cheon and G.T. Leavens. A runtime assertion checker for the Java Modeling Language (JML). Technical Report 02-05a, Iowa State University, 2002.
- [17] A. Clergue. Empirical evaluation of a method for verifying concurrent Java components. M.E. Software Project Report, University of Queensland, 2004.
- [18] S.J. Coakes and L.G. Steed. *SPSS: Analysis without Anguish (Version 10.0 for Windows)*. John Wiley and Sons, 2001.
- [19] J. Cohen. *Statistical Power Analysis for the Behavioural Sciences*. Academic Press, 1969.
- [20] J.C. Corbett. Evaluating deadlock detection methods for concurrent software. *IEEE Transactions in Software Engineering*, 22(3):161–180, 1996.
- [21] B. Curtis. Measurement and experimentation in software engineering. *Proceedings of the IEEE*, 68(9):1144–1157, 1980.
- [22] H. Do, S. Elbaum, and G. Rothermel. Infrastructure support for controlled experimentation with software testing and regression testing techniques. In *Proceedings of the 2004 International Symposium on Empirical Software Engineering (ISESE'04)*, pages 1–11, 2004.
- [23] O. Edelstein, E. Farchi, Y. Nir, G. Ratsaby, and S. Ur. Multithreaded Java program test generation. *IBM Systems Journal*, 41:111–125, 2002.
- [24] S. Edwards, G. Shakir, M. Sitaraman, B.W. Weide, and J. Hollingsworth. A framework for detecting interface violations in component-based software. In *Proceedings of the 5th Intl. Conf. on Software Reuse*, pages 46–55, 1998.
- [25] A. Endres and D. Rombach. *A Handbook of Software and Systems Engineering*. Addison Wesley, 2003.
- [26] Y. Eytani, K. Havelund, S. D. Stoller, and S. Ur. Toward a framework and benchmark for testing tools for multi-threaded programs. *Accepted for Concurrency and Computation: Practice and Experience*, 2005.

- [27] E. Farchi, Y. Nir, and S. Ur. Concurrent bug patterns and how to test them. In *Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS 2003) - 1st International Workshop on Parallel and Distributed Systems: Testing and Debugging*, 2003.
- [28] C. Flanagan and S. N. Freund. Atomizer: A dynamic atomicity checker for multithreaded programs. In *Proceedings of the 31st ACM SIGPLAN-SIGACT*, pages 256–267, 2004.
- [29] B. Goetz. Concurrency in JDK 5.0. In *ibm.com/developerWorks*, 2004.
- [30] H. H. Hallal, E. Alikacem, W. P. Tunney, S. Boroday, and A. Petrenko. Antipattern-based detection of deficiencies in Java multithreaded software. In *Proceedings of the 4th International Conference on Quality Software (QSIC)*, pages 258–267, 2004.
- [31] P. Brinch Hansen. Reproducible testing of monitors. *Software - Practice and Experience*, 8:721–729, 1978.
- [32] K. Havelund and T. Pressburger. Model checking Java programs using Java Pathfinder. *International Journal of Software Tools for Technology Transfer (STTT)*, 2(4):366–381, April 2000.
- [33] A. Hayardeny, S. Fienblit, and E. Farchi. Concurrent and distributed desk checking. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS 2004) - 2nd International Workshop on Parallel and Distributed Systems: Testing and Debugging*, 2004.
- [34] W.C. Hetzel. *An experimental analysis of program verification methods*. PhD thesis, University of North Carolina, 1976.
- [35] G.J. Hidding. Reinventing methodology: Who reads it and why? *Communications of the ACM*, 40(11):102–109, 1997.
- [36] D. Hovemeyer and W. Pugh. Finding bugs is easy. *ACM SIGPLAN Notices COLUMN: OOPSLA onward*, 39(12):92–106, 2004.
- [37] D. Hovemeyer and W. Pugh. Finding concurrency bugs in Java. In *Proceedings of the Twenty-Third Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC 2004) Workshop on Concurrency and Programs*, 2004.

- [38] Sun Microsystems Inc. Programming with assertions. Available online at <http://java.sun.com/j2se/1.4.2/docs/guide/lang/assert.html>, 2002. Retrieved 15 July 2005.
- [39] A. Jedlitschka and D. Pfahl. Reporting guidelines for controlled experiments in software engineering. In *Proceedings of the 2005 International Symposium on Empirical Software Engineering (ISESE'05)*, pages 95–104, 2005.
- [40] N. Juristo and A.M. Moreno. *Basics of Software Engineering Experimentation*. Kluwer, 2001.
- [41] N. Juristo, A.M. Moreno, and S. Vegas. A survey on testing technique empirical studies: How limited is our knowledge. In *Proceedings of the 2002 International Symposium on Empirical Software Engineering (ISESE'02)*, pages 161–172, 2002.
- [42] N. Juristo, A.M. Moreno, and S. Vegas. Reviewing 25 years of testing technique experiments. *Empirical Software Engineering*, 9(12):7–44, 2004.
- [43] N. Juristo, A.M. Moreno, and S. Vegas. Towards building a solid empirical body of knowledge in testing techniques. In *Proceedings of the Workshop on Empirical Research in Software Testing (WERST 2004)*, pages 1–4, 2004.
- [44] N. Juristo and S. Vegas. *Functional Testing, Structural Testing and Code Reading: What Fault Type Do They Each Detect?*, pages 208 – 232. Lecture Notes in Computer Science. Springer-Verlag, 2003.
- [45] E. Kamsties and C.M. Lott. An empirical evaluation of three defect-detection techniques. In *Proceedings of the Fifth European Software Engineering Conference*, Sitges, Spain, 1995.
- [46] T. Katayama, E. Itoh, Z. Furukawa, and K. Ushijima. Test-case generation for concurrent programs with the testing criteria using interaction sequences. In *Proceedings of the 2000 Asia-Pacific Software Engineering Conference*, pages 590–597, 2000.

- [47] B.A. Kitchenham. The case against software benchmarking, keynote lecture. In *Proceedings of The European Software Measurement Conference (FESMA-DASMA 2001)*, pages 1–9, 2001.
- [48] B.A. Kitchenham, T. Dyba, and M. Jorgensen. Evidence-based software engineering. In *Proceedings of 26th International Conference on Software Engineering (ICSE 2004)*, pages 273–281, 2004.
- [49] B.A. Kitchenham, S.G. Linkman, and J.S. Fry. Experimenter induced distortions in empirical software engineering. In *Proceedings of 2nd International Workshop on Empirical Software Engineering (WSESE 2003)*, pages 7–15, 2003.
- [50] B.A. Kitchenham, S.L. Pfleeger, and N. Fenton. Towards a framework for software measurement validation. *IEEE Transactions in Software Engineering*, 21(12):929–944, 1995.
- [51] B.A. Kitchenham, S.L. Pfleeger, L. Pickard, P. Jones, D. Hoaglin, K. El Emam, and J. Rosenberg. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions in Software Engineering*, 28(8):721–734, 2002.
- [52] D. Lea. Overview of package util.concurrent release 1.3.4. available online at <http://gee.cs.oswego.edu/dl/classes/edu/oswego/cs/dl/util/concurrent/intro.html>. Retrieved 12 April 2005.
- [53] D. Lea. The java.util.concurrent synchronizer framework. *Science of Computer Programming*, 58:293–309, 2005.
- [54] R.M. Lindsay and A.S.C. Ehrendberg. The design of replicated studies. *The American Statistician*, 47(3):217–228, 1993.
- [55] B. Littlewood, P.T. Popov, L. Strigini, and N. Shryane. Modeling the effects of combining diverse software fault detection techniques. *IEEE Transactions in Software Engineering*, 26(12):1157–1167, 2000.
- [56] B. Long. *Testing Concurrent Java Components*. PhD thesis, University of Queensland, 2005.
- [57] B. Long, R. Duke, D. Goldson, P. Strooper, and L. Wildman. Mutation-based evaluation of a method for verifying concurrent Java components.

- In *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS 2004) - 2nd International Workshop on Parallel and Distributed Systems: Testing and Debugging*, 2004.
- [58] B. Long and P. Strooper. A classification of concurrency failures in Java components. In *Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS 2003) - 1st International Workshop on Parallel and Distributed Systems: Testing and Debugging*, 2003.
 - [59] B. Long, P. Strooper, and D. Hoffman. Tool support for testing concurrent Java components. *IEEE Transactions in Software Engineering*, 29(6):555–566, 2003.
 - [60] B. Long, P. Strooper, and L. Wildman. A method for verifying concurrent Java components. *Submitted to Concurrency and Computation: Practice and Experience*, 2005.
 - [61] C.M. Lott. Comparing reading and testing techniques. Available online at: <http://www.chris-lott.org/work/exp/>. Retrieved 12 August 2005.
 - [62] C.M. Lott and H.D. Rombach. Repeatable software engineering experiments for comparing defect-detection techniques. *Empirical Software Engineering*, 1(3):241–277, 1996.
 - [63] J. Magee and J Kramer. *Concurrency: State Models & Java Programs*. John Wiley & Sons, 1999.
 - [64] I. Miller and J.E. Freund. *Probability and Statistics for Engineers*. Prentice-Hall, 1965.
 - [65] G. Myers. A controlled experiment in program testing and code walk-throughs/inspections. *Communications of ACM*, 21(9):760–768, 1978.
 - [66] G. Myers. *The Art of Software Testing*. John Wiley & Sons, 1979.
 - [67] N. Nagappan, L. Williams, J. Hudepohl, W. Snipes, and M. Vouk. Preliminary results on using static analysis tools for software inspection. In *Proceedings of the 15th International Symposium on Software Reliability Engineering (ISSRE'04)*, 2004.

- [68] E. Novillo and P. Lu. A case study of selected SPLASH-2 applications and the SBT debugging tool. In *Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS 2003) - 1st International Workshop on Parallel and Distributed Systems: Testing and Debugging*, 2003.
- [69] D.L. Parnas. The limits of empirical studies of software engineering. In *Proceedings of the 2003 International Symposium on Empirical Software Engineering (ISESE'03)*, pages 2–5, 2003.
- [70] D. Perry, A. Porter, and L. Votta. Empirical studies of software engineering: A roadmap. In *Proceedings of the 22nd International Conference on Software Engineering (ICSE)*, pages 345–355, 2000.
- [71] L.M. Pickard, B.A. Kitchenham, and P. Jones. Combining empirical results in software engineering. *Information and Software Technology*, 40(14):811–821, 1998.
- [72] M. Roper, J. Miller, A. Brooks, and M. Wood. Towards the experimental evaluation of software testing techniques. In *Proceedings of EuroSTAR'94*, pages 1–10, 1994.
- [73] D.S. Rosenblum. A practical approach to programming with assertions. *IEEE Transactions in Software Engineering*, 21(1):19–31, 1995.
- [74] G.W. Russel. Experience with inspections in ultralarge-scale development. *IEEE Software*, 8(1):25–31, 1991.
- [75] J. Segal. The nature of evidence in empirical software engineering. In *Proceedings of the Eleventh Annual International Workshop on Software Technology and Engineering Practice (STEP'04)*, pages 40–47, 2004.
- [76] R.W. Selby. Combining software testing strategies: An empirical evaluation. In *Proceedings of the ACM/SIGSOFT IEEE Workshop on Software Testing*, pages 82–90, 1986.
- [77] F. Shull and R. Turner. An empirical approach to best practice identification and selection: The US department of defense acquisition best practices clearinghouse. In *Proceedings of the 2005 International Symposium on Empirical Software Engineering (ISESE'05)*, pages 133–140, 2005.

- [78] D. I. K. Sjoberg, B. Anda, E. Arisholm, T. Dyba, M. Jorgensen, A. Karahasanovic, E. F. Koren, and M. Vokac. Conducting realistic experiments in software engineering. In *Proceedings of the 2002 International Symposium on Empirical Software Engineering (ISESE'02)*, pages 17–26, 2002.
- [79] D.I.K. Sjoberg, J.E. Hannay, O. Hansen, V.B. Kampenes, A. Karahasanovic, N-K. Liborg, and A. C. Rekdal. A survey of controlled experiments in software engineering. *IEEE Transactions in Software Engineering*, 31(9):733–753, September 2005.
- [80] I. Sommerville. *Software Engineering*, chapter Verification and Validation, pages 419–439. Addison-Wesley, 2000.
- [81] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, 1998.
- [82] S. Vegas. Identifying relevant information for testing technique selection. In *Proceedings of the 2004 International Symposium on Empirical Software Engineering (ISESE'04)*, pages 39–48, 2004.
- [83] S. Vegas and V. Basili. A characterization schema for software testing techniques. *Empirical Software Engineering*, 10:437–466, 2005.
- [84] L. Wildman, B. Long, and P. Strooper. Testing Java interrupts and timed waits. In *Proceedings of the Asia-Pacific Software Engineering Conference (APSEC 2004)*, pages 438–447, 2004.
- [85] L. Wildman, B. Long, and P. Strooper. Dealing with non-determinism in testing concurrent Java components. In *Proceedings of the 12th Asia-Pacific Software Engineering Conference*, pages 393–400, 2005.
- [86] C. Wohlin, P. Runeson, M. Host, M. Ohlsson, B. Regnell, and A. Wesslen. *Experimentation in Software Engineering*. Kluwer, 2000.
- [87] M. Wood, M. Roper, A. Brooks, and J. Miller. Comparing and combining software defect detection techniques: A replicated empirical study. In *Proceedings of the 6th European Software Engineering Conference*, 1997.

- [88] M. Young and R.N. Taylor. Combining static concurrency analysis with symbolic execution. In *Proceedings of the ACM/SIGSOFT IEEE Workshop on Software Testing*, pages 170–178, 1986.
- [89] J.H. Zar. Significance testing of the spearman rank correlation coefficient. *Journal of the American Statistical Association*, 67(339):578–580, 1972.
- [90] M.V. Zelkowitz and D.R. Wallace. Experimental models for validating technology. *Computer*, 31(5):23–31, 1998.