Capture-recapture in Software Unit Testing – A Case Study

Hanna Scott
School of Engineering
Blekinge Institute of Technology
SE-37225 Ronneby, Sweden
+46 457 385883
Hanna.scott@bth.se

Claes Wohlin
School of Engineering
Blekinge Institute of Technology
SE-372 25 Ronneby, Sweden
+46 457 385800
Claes.wohlin@bth.se

ABSTRACT
Quantitative failure estimates for software systems are traditionally made at end of testing using software reliability growth modeling. A persistent problem with most kinds of failure estimation methods and models is the dependency on historical data. This paper presents a method for estimating the total amount of failures possible to provoke from a unit, without historical data dependency. The method combines the results from having several developers testing the same unit with capture-recapture models to create an estimate of "remaining" number of failures. The evaluation of the approach consists of two steps: first a pre-study where the tools and methods are tested in a large open source project, followed by an add-on to a project at a medium sized software company. The evaluation was a success. An estimate was created, and it can be used both as a quality gatekeeper for units and input to functional and system testing.

Keywords
Unit test, capture-recapture, prediction, faults, failures

1. INTRODUCTION
Capture-recapture was initially a method applied in biology. It has then been transferred to software engineering to be used in software inspections and testing. In biology, capture-recapture is used to estimate animal populations, for example, the number of deer in the woods, and it is also used in medical research [1]. In software engineering, capture-recapture methods have been used for fault seeding methods [2]. By seeding fault, and comparing the number of faults found that was seeded and non-seeded faults, the number of remaining faults can be estimated. Later, capture-recapture was applied in software inspections to estimate the number of remaining faults. The first application of capture-recapture on software inspections was published in 1992 by Eick et al. [3]. A survey of the capture-recapture in software inspections is provided in [4]. Capture-recapture has also been applied in different ways in testing [5, 6].

In software engineering, capture-recapture can be described as a method that uses the overlap between inspectors or testers to estimate the remaining number of faults. The overall hypothesis is that if, for example, several testers have a large overlap in what they find then few faults are probably remaining. A similar reasoning can be used if the overlap is small between the different testers. Different statistical methods can be used to make the estimation.

To the best of our knowledge, capture-recapture has not been applied in software unit testing in the way reported here. The application of capture-recapture in unit testing has two objectives. First of all, it can be used for quality control. In other words, as a gatekeeper to ensure that whatever we pass on to the following test phases is of sufficient quality. Second, it can be used as a method to try to estimate the remaining number of faults as such.

Thus, the objective of this paper is to illustrate through an industrial case study how capture-recapture is possible to apply to software unit testing to obtain an estimate of the remaining number of faults. The design of the case study is presented in detail together with findings from a pre-study conducted on an open source project and the main study conducted at a company.

The study shows that it is possible to estimate the number of remaining faults. The additional costs for introducing parallel testing of software units are reported. The study provides input to companies to judge whether parallel unit testing is a way forward to get a better hold on the software quality early in the testing activities.

The paper is structured as follows. Section 2 presents related work. The case study is presented in Section 3, where the context, design, execution, threats and results are presented. A discussion is provided in Section 4. Finally, Section 5 presents the main conclusions from the study.

2. RELATED WORK
The related work of this paper contains two main sections. The information and research on combining unit test results with capture-recapture is sparse, hence the related work starts with the use of capture-recapture in software inspections, then moves on to related works in capture-recapture applied to unit tests.

2.1 Capture-recapture in inspections
Capture-recapture estimations are based on the assumption that the overlaps between the number of animals captured the first,

Conference'04, Month 1–2, 2004, City, State, Country.

second and third time are inversely proportional to the amount of animals not captured at all. The fewer the number of animals in the overlap, the larger the number of animals not captured at all.

The main difference of applying capture-recapture in software inspections, compared to biology, is the data collection method. The method is serial in biology and simultaneous in inspections.

There are four main models to use for estimation as part of the capture-recapture methods usually referred to as capture-recapture models M0, Mh, Mt and Mth [4]. The differences between models are the basic assumptions of how the world works, and in particular the probabilities of faults being detected and the ability of reviewers to detect them [4]. The four capture-recapture models' with varying assumptions are:

M0 assumes that the probability of a fault being found is the same for all faults, and that the ability of the inspectors to find each fault is also the same.

Mh also assumes that the probability of a fault being found is the same for all faults, but it assumes that the detection ability can vary from inspector to inspector.

Mt here the assumption is that the probability of faults being found varies, but that the inspectors all have the same ability to find each fault.

Mth makes neither an assumption about probabilities being the same for faults nor for inspectors, i.e. the probability of faults being found can vary, and the ability to find each fault can vary from inspector to inspector.

When using a capture-recapture model in practice, the model needs to be complemented with an approach to predict the number of remaining faults. The approach, or formula, is generally called an estimator [4]. The three most common estimators today are: Maximum likelihood (ML), Chao's estimator (Ch) and Jackknife (JK). The combination of model and estimator that has had the greatest success in software inspections is Mh with the Jackknife estimator (Mh-JK) [4]. There are different orders of estimators for the Jackknife method and according to both Miller and Thelin et al. the second order Jackknife estimator performs best [7].

An overview of how Mh-JK has performed in the past on software inspection data is presented in Table 1. The results are presented as Mean Magnitude of Relative Error (MMRE) measurements in order to give a better overview. The MMRE measurement has a tendency to underestimate in comparison to the mean, and it also has a high probability of selecting a model that has a bad fit to the data [8], but since most papers in the area were written before 2003 when the critique of MMRE was published the contents of Table 1 still uses this measurement. Usually a combination of model and estimator is considered reasonably accurate when it has a bias in the approximate range of ±20% [9, 10].

Thelin's [7] results in Table 1 were not originally written as an MMRE measurement but in order to be able to compare the results with that of Petersson et al. [11] and Briand et al. [10] the data available in [7] has been transformed into MMRE. The results in [10] and [4] are available as MMRE.

Current work in the area of applying capture-recapture on software inspection data mainly consists of comparing the method to other methods of estimation [4]. There is still no agreement on whether or not the capture-recapture method outperforms subjective estimates, but the two methods seem to have similar performance [7, 11]. There is also an interesting trend of evaluating the stability of estimations using the capture-recapture method [12, 13].

Table 1. Example of performance of Mh-JK in inspections

Researchers	No. of Insp.	No. of subjects	MMRE in %
Thelin [7]*	3	34 BSc stud	-16
Petersson et al. [4]	3	Simulation	19
Thelin [7]*	3	48 MSc stud	21
Petersson et al. [4]	4	Simulation	28
Briand et al. [10]	3	14(13) professionals	28
Briand et al. [10]	2	12 professionals	-52
Petersson et al. [4]	4	Simulation	-64
Petersson et al. [4]	3	Simulation	-72

2.2 Capture-recapture in testing

When applying the capture-recapture method on failure data from testing there is one thing that needs to be kept in mind: the artifact, being the code in execution, changes faster than other artifacts. In order to give a representative picture of the test process it is important that failures of type "test stoppers" are allowed to be removed. This means that the artifact being scrutinized is being changed while the testing is being done. Since the code is in execution and works together with other programs and underlying platforms, a failure can be caused by factors other than the artifact under scrutiny. Thus, the best combination of capture-recapture model and estimator may very well be different in testing than the ones found to be most suitable in different inspection studies.

There are few results from applying capture-recapture to test data, but there are at least two, one by Stringfellow et al. [5] and one by Yang et al. [6]. In the study by Stringfellow et al, the objective was to estimate which units that can be expected to contain faults in operation even if faults were not found in testing. In other words, the study was not focused on estimating faults as is the goal in our paper. The results from the Stringfellow et al. analysis show that the capture-recapture model and estimator combination seems different, even if the Mh-JK estimator is within acceptable range in testing as well as in inspections [5]. The capture-recapture models that perform within an acceptable level of the subjective estimates in the Stringfellow et al. study are: Mt-ML, M0-ML and Mh-JK [5]. Given that the objective here is not to compare different capture-recapture models and estimators, a selection of models and estimators to use had to be made. As these three estimators have performed well previously when used with test data, these estimators are used in this study and therefore require some additional explanation. Furthermore, at least the Mt-ML and Mh-JK models and estimators have been among the most successful in inspections studies as well [4]. Another similarity to inspections is that the subjective estimates perform very well, but that the capture-recapture models seems to have similar performance to the subjective estimates [5].

The Yang et al. [6] investigation tries to determine when to stop testing through estimating "defects" found in test using, among

other techniques, recapture debugging. Recapture debugging is built on the existence of reported duplicates in testing, and different failures that can be traced back to the same root cause. Yang et al. look at examined "defect" capture-recapture, i.e. they investigated the root causes of the failures instead of looking at the failures themselves as separate occurrences [6]. The study found that using capture-recapture to determine when to stop testing is only superseded by the optimal test stop criteria, which is defined as "when it becomes cheaper to fix a defect in maintenance than in test" [6].

3. CASE STUDY

The case study presented in this paper aims to evaluate if it is possible to use capture-recapture to estimate the number of remaining faults after unit testing. The case study is conducted by having several testers performing unit testing on the same unit, ideally in parallel. In other words, testing is conducted in a similar way as individual reading in inspections. The case study is conducted as an extra test activity. This is done for two main reasons. First of all, this is a pilot study and hence it is not integrated into the normal development process. Second, the study is conducted after the unit test prescribed in the development process is done. The latter has one major advantage. The quality of the unit is higher, which means that failures of type "test stoppers" have been removed and several testers do not have to find the same "simple" failures.

It is important to point out that our application of capture-recapture in unit testing has several different goals. First, the company should evaluate whether the use of capture-recapture is a useful step to assure quality of individual units and to help in planning of the remaining test phases based on the failure estimates. Second, even if the company decides not to use this approach for all units, it will be evaluated whether it can be applied for some units to obtain an estimate of the total number of remaining failures in a subsystem. The latter requires that a representative unit is selected, i.e. a unit that is estimated to have an average number of failures in relation to the other units in the subsystem. This objective resulted in an extra step in the case study, since capture-recapture study should be applied to a "representative" unit.

The presentation of the case study is divided into five parts: context, design, execution, validity and result of the study.

3.1 Context

This case study was made in two different environments. In the pre-study, the environment is the Open Source project called KDE. The pre-study was presented in a news article for the KDE community, and anyone could sign up for the study. The KDE community is a voluntary-based development community, which usually means that people will participate in the activities they enjoy. The voluntary basis also meant that among the people who signed up for the pre-study and participated in answering the pre-test questionnaire, only 7% chose to participate in the actual test part of the study. This is further discussed in Section 3.3.1.

The main study was conducted at a medium sized software company named UIQ Technology, with a total of 366 employees, of which 98 are developers. The company develops Symbian C++ software for mobile phones, so a lot of their software is aimed at a very diverse mass market. The development is in the process of being phased from traditional waterfall development approach to incremental software development process applying test-driven development. UIQ Technology employs outsourcing on a regular basis, partly because they are investigating if outsourcing can save resources, and partly since the demand for mobile software is increasing there is a shortage of software developers in Sweden today. The outsourced software is usually one component from the baseline maintained at the company. This case study was performed at UIQ Technology because it was agreed that it would help the main organization if they could tell how many faults the outsourced components contain when they arrive for integration into the baseline. In other words, there was a high demand for finding ways to obtain earlier estimates of software quality, and in particular to have a better control on the quality of components that are part of the outsourcing.

3.2 Design

This section contains a description of the case study design for applying capture-recapture to failure data from unit tests. The study design is divided into two phases with eight distinct steps each. The first phase is a pre-study and the second phase is the main study. The pre-study is conducted to evaluate the steps and to ensure that the likelihood for success of the case study at the company is maximized. The eight steps of the design are:

1. Select a unit for testing
2. Pre-test questionnaire handout to developers
3. Selection of four developers
4. Parallel unit test execution
5. Participants' subjective estimates
6. Meeting to define failures and overlap
7. Capture-recapture calculations on representative unit
8. Inform all involved of study results.

Each design steps is described in more detail in the following subsections.

3.2.1 Select a unit for testing

In order to find a unit with failure representative traits the units need to be measured. This means that a data extraction is needed followed by an analysis of the data. A number of metrics are extracted from two releases of the same software product. The intention is to identify measurements that correlate with number of provoked failures from an old release of the project, and then use the same measurements in the current release to identify a "representative" unit to use in the capture-recapture study. This is done by using the failure correlation measurements from the old release together with the metrics measurements from the new release to create a representative conceptual unit with the median values of all measurements. The unit with values closest to the conceptual unit is then selected for being used in the study.

3.2.2 Pre-test questionnaire handout to developers

A list of potential participants in the study is created. The people on the list are asked to complete a questionnaire. The questionnaire asks about their programming and testing skills, as well as motivation level. This step is done to ensure that the subjects chosen for the study are representative of developers at the company.

3.2.3 Selection of four developers

The answers from the questionnaire are used to determine the median skill level of the entire group, and select four developers in the immediate vicinity of the median to perform the parallel testing. The questionnaire answer options are Likert scale alternatives. The analysis consists of prioritizing the questions according the believed impact of the answers to the parallel testing. For each question, the most selected answer option is considered the median value of the group. When selecting suitable participants for the study, the priority of a question is taken into account. The higher the priority of a question, the more important it was the developers selected had provided an answer that was equivalent to the median of the group.

It is important to point out, that the people, who already had expertise or experience with the component to be tested, were excluded from the group of potential participants to avoid bias in the results.

The four developers are briefed about the details of the study a few days before the parallel test session via e-mail, or if possible, face-to-face. The briefing includes: a description of goal of the capture-recapture study and their role in the study. The potential gains are also described.

3.2.4 Parallel unit test execution

The four developers are expected to test the same unit. The testing is done in parallel. While performing the tests they are to note activities, activity switches, failures found, time each failure was found, how each failure was provoked and the symptom of each failure.

In order to make sure the unit testing is going as planned the main researcher collects verbal, individual feedback from the participating developers three hours into the unit test. This could potentially cause changes to the design during execution of the case study.

3.2.5 Participants' subjective estimates

At the end of the parallel tests, each of the four developers provide his or her subjective estimate of how many more failures they think they would be able to provoke if given as much additional test time as they wish. This is viewed as an estimate of the remaining number of failures in the unit.

3.2.6 Meeting to define failures and overlap

The four developers have a meeting with the main researcher performing the study (main author of the paper), where the developers discuss all "failures" noted during the parallel tests, and then decide what constitutes a failure. In this meeting, the overlap matrix for the failures is determined. At the end of the meeting the participants perform an open-ended qualitative evaluation of the study's different aspects.

3.2.7 Capture-recapture calculations on representative unit

The main researcher performs the capture-recapture estimations for all combinations of all possible groups. The models used are M0, Mt and Mh, and the estimators are both Maximum Likelihood and Jackknife. The combinations used in this study are the ones that showed promise in the Stringfellow et al. study [5]: Mt-ML, M0-ML and Mh-JK. Note that the Mh model assumes that the detection ability of the developers are different, and the developers are selected based on the median. The use of the Mh model is still relevant, because the pre-test questions might not be able to catch what constitutes different detection abilities between developers.

3.2.8 Inform all involved of study results

The results of the study should be presented at a meeting with the entire group of possible participants, i.e. the people who participated in the testing and their managers. This step is important in many ways. It gives the company representatives a sense of participation, and it explains to them what happened to their invested hours. Furthermore it increases the visibility of the researcher at the company for future investigations and it is a first step toward creating a company opinion about the approach.

3.3 Execution

The execution of the study is reported in several steps. In the next section, the execution of the pre-study is reported. The following sections describe each of eight steps in the design for the main study.

3.3.1 Pre-study of the capture-recapture study

The study execution was preceded by a pre-study, made on the KDE Open Source project. The pre-study was intended to encompass the eight steps in the design for evaluation purposes. An invitation was sent out and the interest for the study was much higher than expected among the developers of KDE. The study started off with 41 developers answering the pre-test questionnaire on skills. Out of the 41 developers who answered the pre-test questionnaire, only three handed in a result from the unit testing part of the study. The three results handed in for analysis did not contain any overlap, hence the pre-study could not produce an estimate since some overlap is a prerequisite to make the capture-recapture models work (in terms of producing an estimate). There was a qualitative evaluation on what went wrong among all 41 initial participants. Among the developers who did not hand in their unit test package, nine gave qualitative feedback on the following questions:

a) Why did you not proceed /finish the unit test part of the study?

b) Do you have any other general feedback on the pre-study?

The feedback clearly showed that all the developers could not find enough time for the five-hour time slot needed to perform the unit test. Four developers also stated that they ran into scheduling problems, due to that the date for the study was moved two weeks. The move was due to difficulties finding someone to write documentation for the selected unit to test. Two developers stated that they simply realized they did not have enough skills to complete the unit tests.

The three developers, who did hand in their unit test package, were asked the following questions:

a) Why do you think so few of the initially interested participants did not proceed/finish the unit test part of the study?

b) What parts of the unit test parts of the study do you think could be improved and how?

The answers from the developers showed that the main problem with the study was the time it took to set up the test environment and create their own test harness from scratch.

As a direct result of the test run of the capture-recapture pre-study on the KDE Open Source project, the tasks of "Write code documentation and test driver template" were added to the design step "Selection of four developers".

3.3.2 Select a representative unit.

The experiences from the pre-study were important inputs to the main study. The first step was to find a representative unit, all units first needed to be measured to get an overview of the unit set of the entire outsourced subsystem. The main researcher performed the data extraction. The approach was determined over a six-month period of time. This created some challenges since projects are very dynamic and in this particular case the project, and hence the potential units to use, were changed three times. In the end, data was extracted from one old subsystem of the system delivered from the subcontractor selected (the development was outsourced), and from a new subsystem from the same outsource company that had just been delivered. The measurements collected were:

- Number of times the cpp files was updated (cpp denotes a code file that contains the implementation of functionality)
- Number of times the h files was updated (h denotes a header code file, meaning the file where the functionality for the cpp files are declared)
- Source Lines of Code (SLOC) in cpp
- Commented Lines of Code (CLOC) in cpp
- Simple outbound function calls (MPC)
- Simple inbound function calls
- Complex indirect outbound function calls
- Complex indirect inbound function calls
- Number of C++ classes in the unit
- Number of functions in the C++ unit
- Number of members in the C++ unit
- SLOC in header files of the unit
- CLOC in header files of the unit
- Number of types in calls

The old subsystem had 28 units, with only three failures reported to two units. This meant that we were unable to determine how well or badly the number of failures correlated with these measurements. It is, however, known that all of these measurements have shown to be highly correlated with the number of faults in other research studies [14,15], so we still created a conceptual unit based on the median value of all measurements. The unit with the closest actual values to those of the conceptual unit was chosen for the study.

Finding a representative unit took about eleven hours. These eleven hours include analysis of the data collected in the form of normality tests and correlation analysis. The time does not include the overhead of setting up and using parsing tools or re-extraction time when the projects were changed etc.

3.3.3 Pre-test questionnaire handout to developers

The pre-test questionnaire was handed out to 27 developers in total. The lead-time for collecting all questionnaires was about a month, because some developers went vacation, others came back from vacation, and some got sick.

One of the developers was not on site during when the first batch of questionnaires were handed out, so he answered the questionnaire about a week after the rest of the development department.

The two last developers to perform the unit tests answered the questionnaire about one month after the rest of the group of developers.

The spontaneous feedback given by about a fourth of the developers who answered the questionnaire was that some of the questions were hard to understand. It was however judged that these difficulties did not affect the actual selection in the next step.

3.3.4 Selection of four developers

Some problems were found when doing the analysis of the answers from the questionnaire. The answers revealed that the company had two large and very different groups of developers when it comes to skills and motivation. It should be noted that the motivation factors of the pre-test only refer to the willingness to participate in the study. Since the company had two different skill and motivational groups, half of the candidates were selected from each group. The candidates were selected based on the most frequently occurring answers to the questionnaire questions, within each skill and motivational group.

In order to avoid having to redo the selection again, in case some of the developers' chosen were assigned to other departments temporarily or such, the selection contained six primary candidates, and four secondary candidates.

In the end, it was discovered that two out of the first four candidates were not able to perform the tests because a vital skill required to carry out the tests was forgotten in the pre-test questionnaire. In the end this meant that two out of the four developers whose results qualified were not selected using median answers on the pre-test questionnaire. The last two developers were chosen based on their pretest but with the additional information on their knowledge of a specific tool used in the unit testing. This is further discussed in the validity section under 3.4.

Based on the outcome of the pre-study, two new tasks were added in comparison to the initial design of the study. The added tasks were:

Task 1: Write code documentation and test driver template

- The code documentation on the selected unit should be complemented where needed. This is important since a failure is a deviation from the required behaviour hence for the developers need to understand as exactly as possible the intended behavior of the unit being tested.
- A test driver needs to be created for the unit test session. The template should be empty, but contain all dependencies on needed code units that will allow the code to be run, but nothing more in order not to guide developers too much in their testing.

Task 2: Set up of test environment:

There should either be extra time for the developers to set up their test environment or make sure there is another person who can set up the test environment quite quickly, and answer any questions regarding it that the developers might have. For this case study the person responsible for the integration and test of the component was called in to set up the test environment.

3.3.5 *Parallel unit test execution*

The parallel unit testing was divided into two sessions. The first two developers carried out the first unit test session about two weeks after the pre-test questionnaires had been handed in. The two last developers performed the second unit test session about two weeks after the first session. The sessions were carried out in the same way. The subjects in the study were informed about the necessity not to share any experiences until all four developers had performed the unit test. Before the test, the developers gathered for one hour information meeting in the morning, and then they had seven hours to perform the testing. Three hours into the unit tests execution, the main researcher collected feedback from the participants on how the testing was proceeding. The feedback from the developers stated that they were having problems understanding the dependencies of the unit. This caused a change to a different test harness when two hours of unit tests remained. The first test driver was provided from the beginning of the testing sessions, and it was empty so they could fill it up themselves. A second test driver template was provided when two hours of the test remained due to feedback from the participants. The developers said would be much easier to extend already existing flow test cases. The implications of adding a second test harness was considered for two hours, because it was vital not to guide the developers too much during the unit testing. It was decided it was worth the risk since the addition could at most cause the developers to miss failures caused by non-allowed execution sequences, and it was not possible for end users to manipulate the unit in this way.

The outsource company that developed the subsystem used in the study provided the latter test harness when the code for the subsystem was delivered. The test harness consisted of extendable basic flow tests.

3.3.6 Participants' subjective estimates

Two of the four developers provided subjective estimates of the number of remaining failures. One developer estimated four failures to be left, and the other one estimated five failures to be left. In both cases this meant that they estimated a total failure contents before test to be six failures.

3.3.7 Meeting to define failures and overlap

This meeting went according to plan. The meeting took about one hour, and included: a discussion on which of the reported failures could actually be seen as failures, defining overlaps among the found failures and lastly, general, unstructured qualitative feedback on the entire study up until this meeting. The results of the overlap analysis can be seen in Table 2.

The qualitative feedback from the developers on the study was as follows:

- The documentation of the code was an insufficient basis on which to specify test cases from scratch. The developers all agreed that if the documentation is good enough, the meeting for defining the overlap might not be necessary.
- The documentation in the flow test case harness provided during the last two hours of test, had too poor documentation. The developers suggested the use of Doxygen [17] and Docbook [18] style documentation.
- The empty test harness provided from the beginning of the test sessions contained bugs, so for future studies or actual use of the approach the test harnesses should be tested properly before the test sessions begin.
- A set-up description of the test environment should have been included. Even though the developers had used the test framework before, setting up the correct test conditions for the specific unit still took 2-3 hours for each of the developers.
- There was a suggestion that a time period would be set aside for the participants, during which they could focus on getting familiarized with the code, without feeling pressure to start testing.

3.3.8 Capture-recapture calculations on representative unit

The capture-recapture calculations were performed in Matlab. The results can be seen in Table 3.

3.3.9 Inform all involved of study results

The meeting was conveyed to the team leaders of the teams that provided the developers. The developers themselves were invited to the meeting but chose not to participate. The study results were well received, and now the company is considering implementing a large-scale pilot evaluation of the method.

3.4 Validity

The validity of the case study presented in this paper can be viewed in two different ways: design validity and validity of the results.

The adaptation of the design done at the execution stage to fit reality at the company that could have an effect on the validity of the results can be summarized as follows:

- The change of selection for the last two developers according to an additional criterion may be a validity threat to the generalization of the applicability of the approach within the software company itself.
- There were no resources available to improve the documentation on the unit selected for testing so the testing was performed on code with partially incomplete documentation. This may have resulted in fewer failures being detected than if the documentation would have been better.
- Because of a misunderstanding of the term "unit test" in the pre-test questionnaire, two of the four developers originally selected were found not have the experience needed to perform the unit testing, hence only two out of four selected developers could produce a result which qualified for the estimate calculations. To resolve the problem two additional developers were selected to do the unit test. A second unit test session was scheduled. The second session took place about two weeks after the first session, where the participants' experience of unit tests had been established before hand. After the first test session, the participants of that session were instructed not to discuss their findings or the study with other developers at the company until after the

second test session had been completed. However, whether or not this rule was followed cannot be known.

Performing a pre-study has increased the validity of the case study design presented in Section 3.2. However, some additions and changes to the design is part of the outcome of the main case study as well. These are the changes, which should be made to the design before the study is run again:

- The pre-test questionnaire needs to be modified.
- The flow test case harness will be used from the beginning of the test session. The addition was a result of feedback from the developers while the unit testing was being done, and it showed a much higher rate of failure provocation than the use of the first, empty test harness.
- A description on how to set up the test environment should be included in the test kit.
- The requirement on the participants to provide a subjective estimate of remaining failures at the end of unit testing needs to be stated more clearly.

3.5 Results

The outcome of the capture-recapture study is presented in Table 2. In total, the four developers (denoted D1 to D4) found five failures in unit testing (denoted F1 to F5). Having failures marked by "X" in grey cells shows failures found by the reviewers. It is somewhat surprising to see the relatively small overlap. For example, even if D1 and D2 together find all five failures identified in the study, there is no overlap between them. The data in Table 2 can be used to estimate the total number of failures and in particular the number of remaining failures using the capture-recapture models and the estimators listed in Section 2.

From Table 2, it is possible to make estimates based on two, three or four developers. From a cost perspective, it would be preferable if as few developers as possible are used. However, it is of course important to find as many failures as possible too and to be able to make trustworthy estimations. For example, on the one hand D1 and D2 find five failures together, which is very good. On the other hand, it is impossible to make an estimate and the lack of overlap between them indicates that several failures are remaining. The estimates for all possible combinations of developers using different capture-recapture models and estimators are shown in Table 3.

Table 2. Overlap of the failures found in the unit test session

	D1	D2	D3	D4
F1	X		X	
F2	X			
F3		X		X
F4		X		
F5		X		

The first column shows the different combinations of developers where it is possible to get an estimate. Some overlap is required to get an estimate. The estimates in Table 3 show an estimate of the total number of failures, i.e. including those found in the unit test. It can be noted that the estimates for two developers (D1, D3 and D2, D4) are low. This is due to that estimations are only possible for two groups with no overlap between them. In other words, developers D1 and D3 together only found two failures and developers D2 and D4 only found three failures. When moving to three developers, the estimates become higher and closer to the estimate obtained for four developers. The only exception is the estimate for three developers when developer D2 is excluded.

Unfortunately, the actual number of failures is unknown and hence the estimates cannot be compared with an actual value. The estimates shown in Table 3 are for the three different models with different estimators. The models are rather consistent and looking at the estimates it is reasonable to state that the total number of failures in the unit is 7-8 failures, which means that it is estimated that 2-3 failures are remaining after the study. 7-8 failures are considered the result of the estimation because the median values for the groups containing three or four developers are 7.23 and 7.67. The groups of three to four developers were used as the primary estimators of the actual number of failures, since the more developers the more reliable the result.

The estimates from the models could be compared with the subjective estimates provided by two of the developers. Both these developers estimated the total number of failures to be six. Thus, the estimates of the models and developers are close to each other, which provides some support for the results of the estimations.

Unfortunately, it was impossible to compare the estimates from the capture-recapture models and the subjective estimates with the actual outcome, since this information is not available.

Table 3. Capture-recapture model estimate results for all possible group combination

Testers	Mt-ML	M0-ML	Mh-JK1	Mh-JK2	Mh-JK3
1,3	2	2	2.5		
2,4	3	3	4		
1,2,3	8	9	7.67	8.83	
2,3,4	4	6	6	6.83	
1,2,4	8	9	7.67	8.83	
1,3,4	3	3	4.33	4.83	
1,2,3,4	6	6	7.25	8.08	8.83

4. COMPANY IMPLEMENTATION

Applying capture-recapture to unit testing provides two main possible uses. First of all, the estimates could be used as quality control, i.e. as a gatekeeper to decide whether a specific unit could be transferred to forthcoming test phases or if more unit-testing is needed, or even if the unit should be sent back to development. Second, the estimate provides information about the potentially remaining number of failures and hence this information can be used to plan testing activities. It can be used to direct resources both in terms of person-hours and where in the system to put most testing resources.

In this particular case, the main organization gets information about the quality of the outsourced software that it did not have before. The estimate provides information that can be used to decide whether or not to integrate the component into the baseline at the current quality level it exhibits.

An important question for the company is the actual cost in comparison to the potential gains. In this case the main researcher conducted some of the work. However, the case study did incur some costs for the company too. The costs for the company are shown in Table 4, where it can be seen that the total cost was 51 hours.

The developers estimated that the use of the second test harness would allow the testing to be done in just four hours per developer. An analysis of the results from a budget perspective was conducted within UIQ Technology. The analysis showed that, if the failures found are in need of corrective action, the cost of the study is outweighed by the time saved on system testing later on in the project. The results were calculated for the UIQ context, using four hours of testing per developer, and the amount of failures being the same as in the study. For UIQ Technology, it means that they would get the additional information of the estimation without any additional costs to the development projects.

Table 4. Company investment for the add-on

Time in hours	Activities
10	Person hours for all developers to fill in the questionnaire
1	Creation of first test harness
2	Developer to help set up test environment
6	Meeting to prepare developers for unit test
28	Developers performed unit tests
4	Hours to determine failure overlaps and evaluate the study
Total 51	Company investment

In addition, the researchers invested time in the case study. The total number of hours is shown in Table 5.

Table 5. Transferable tasks

Time in hours	Activities
80	Extraction of all module measurements
1	Selection of the module to test
8	Coordinating the questionnaire collection.
14	Selection of the developers to perform the unit tests
9	Coordinate the unit tests
1	Coordinating the evaluation meeting
1	Performed capture-recapture estimate calculations
Total 106	All activities, which could be transferred from the researcher to the company.

It is important to stress that the largest cost is related to the extraction of measurements, which is needed to select a representative unit. This cost only occurs if the aim is to identify and representative unit. If the intention is to perform this type of quality control on all units then the costs in both Tables 4 and 5 have to be adapted to this. In other words, it should be remembered that the costs here are for one unit. However, the costs may become lower as we learn how to conduct these types of studies.

In order for UIQ Technology to get general use of the method, the tasks above have to be done by company personnel. The extraction of measurements was done manually due to certain features of the programming language that made automation difficult. The extraction can be automated, but at a cost of around two full time weeks of implementation. The same automation took about two full weeks to implement for C++ in the pre-study.

4.1 Use of results

The estimate produced can be used as a basis to make the decision to integrate the outsourced component or not. The use of the method is not limited to outsourced components, but could potentially be used to estimate the "failure contents" of in-house components as well. The method could also be used on components before release to the customer, in order to sample the quality level of the final product. If applied just before delivery, the resulting estimate could help determine the size of the project's maintenance budget, if the sampled component could be proven to be representative for the system in some sense.

5. CONCLUSIONS

The contribution of this paper is the unique combination of the results from unit tests with capture-recapture calculation models. The results of this case study were positive. The method's different parts were evaluated for applicability on a real software project. The capture-recapture on unit tests is a method that is applicable in a commercial software company environment. The method is very cost efficient in the sense that it both finds failures and provides information to make an estimate of the remaining number of failures in a unit. It is however very important not to perform the unit testing to early, to avoid "show stopper" faults. In the case study, this was secured by performing the parallel unit testing after the unit test included in the development process at the company. An estimate was created from this case study. The resulting estimate is comparable with the subjective estimates of the participants. The estimate itself can be used to control the actual quality of a unit and as information for further planning of testing activities.

ACKNOWLEDGMENTS

Special thanks to Dr. Thomas Thelin for providing the Matlab files for the calculations.

We would also like to thank UIQ Technology staff who made this study possible.

A big thank you to the KDE project for the support they gave the pre-study, and especially Till Adam and Matthias Kalle Dahlheimer at Klarälvdalens Datakonsult AB (KDAB).

This work was partly funded by The Knowledge Foundation in Sweden under a research grant for the project "Blekinge - Engineering Software Qualities (BESQ)" (http://www.bth.se/besq).

REFERENCES

[1] Anne Chao, 1998. Capture-Recapture Models, Encyclopaedia of Biostatistics, Editors: Armitage & Colton, Wiley, New York.

[2] Harlan D. Mills, 1972. On the Statistical Validation of Computer Programs, Technical report FSC-72-6015, IBM Federal Systems Division.

[3] Stephen G. Eick, Clive R. Loader, M. David Long, Lawrence G Votta, Scott A. Vander Wiel, 1992. Estimating Software Fault Content Before Coding, In: Proceedings of the 14th International Conference on Software Engineering, 59-65.

[4] Håkan Petersson, Thomas Thelin, Per Runeson, and Claes Wohlin, 2004, Capture-recapture in software inspections after 10 years research--theory, evaluation and application, Journal of Systems and Software 72,2 (July, 2004), pp. 249-264.

[5] Catherine Stringfellow, Anneliese Andrews, Claes Wohlin and Håkan Petersson, Estimating the number of components with defects post-release that showed no defects in testing, Software Testing, Verification and Reliability 12,2 (2002) pp. 93-122.

[6] Mark C. K. Yang and Anne Chao, Reliability-estimation and stopping-rules for software testing, based on repeated appearances of bugs, IEEE Transactions on Reliability 44,2 (1995), pp. 315-321.

[7] Thomas Thelin, 2004, Team-based fault content estimation in the software inspection process, Proceedings of the 26th International Conference on Software Engineering (Edinburgh, Scotland, United Kingdom, May 23 – 28, 2004), 263-272.

[8] Tron Foss, Erik Stensrud, Barbara Kitchenham, and Ingunn Myrtveit, 2003. A simulation study of the model evaluation criterion MMRE, IEEE Transactions on Software Engineering, 29,11 (Nov. 2003), pp. 985-995.

[9] Amr Kamel, and Paul G. Sorenson, The application of capture-recapture log-linear models to software inspections data, Proceedings of the International Symposium on Empirical Software Engineering (Rome, Italy, Sept. 30 – Oct. 1, 2003), 213-222.

[10] Lionel C. Briand, Khaled El Emam, Bernd G. Freimut, and Oliver Laitenberger, A comprehensive evaluation of capture-recapture models for estimating software defect content, IEEE Transactions on Software Engineering, 26,6 (June 2000), pp. 518-540.

[11] Håkan Petersson, and Claes Wohlin, An empirical study of experience-based software defect content estimation methods, Proceedings of the 10th International Symposium on Software Reliability Engineering (Boca Raton, Florida, United States, Nov. 1 - 4, 1999), 126-135.

[12] Ching-Pao Chang, Jia-Lyn Lv, and Chih-Ping, Chu, A defect estimation approach for sequential inspection using a modified capture-recapture model, Proceedings of the 29th International Computer Software and Application conference (Edinburgh, Scotland, United Kingdom, July 25-28, 2005) vol. 1, 41-46.

[13] Thomas Thelin and Per Runeson, Confidence intervals for capture-recapture estimations in software inspections, Information and Software Technology 44,12 (Sept. 2002) pp. 683-702.

[14] Lionel C. Briand, John W. Daly, Jürgen K. Wüst, A Unified Framework for Coupling Measurement in Object Oriented Systems, IEEE Transactions on software engineering, 25,1(January 1999), pp 91-121.

[15] Norman E. Fenton, Martin Neil, Software metrics: roadmap, Proceedings of the conference on The Future of Software Engineering, (Limerick, Ireland, June 04-11, 2000), 357-370.

[16] Dimitri van Heesch, Doxygen, http://www.stack.nl/~dimitri/doxygen/ 2008-02-29.

[17] Dokbook, http://www.docbook.org/ 2008-02-29.