Intelligent Information Management, **2010, 2, 343-353**
doi:10.4236/iim.2010.26042 Published Online June 2010 (http://www.SciRP.org/journal/iim)

Status of Developers' Testing Process

Gudrun Jeppesen[1], Mira Kajko-Mattsson[2], Jason Murphy[3]
[1]*Department of Computer and Systems Sciences*, *Stockholm University*, *Stockholm*, *Sweden*
[2]*School of Information and Communication Technology*, *Royal Institute of Technology*, *Stockholm*, *Sweden*
[3]*Nomadic Software*, *Nomadic City*, *Sweden*
E-mail: *gudrun@dsv.su.se*, *mekm2@kth.se*, *jasonleemurphy@hotmail.com*
Received March 12, 2010*; revised April* 15, 2010; *accepted May* 17, 2010

Abstract

Even if recent methodologies bring more recognition to developers' testing process, we still have little insight into its status within the industry. In this paper, we study the status of developers' testing process at Nomadic Software. Our results show that the process is not uniformly executed. The company suffers from lack of control over the methods used, lack of formal communication on requirements, lack of static testing practice, and lack of testing process documentation.

Keywords: Dynamic Testing, Static Testing, Peer Reviews, Inspections, Debugging, Test Cases, Testing Techniques

1. Introduction

Despite its importance, the overall testing process has for many years been neglected both within research and industry [1-3]. Most of the effort has been spent on creating testing processes on the system level. Hence, we have fairly good understanding of system testing and its industrial status. Regarding the other levels, such as unit (developer level testing), integration and acceptance testing, little, if almost nothing, has been done both within the academia and industry.

Recently, developers', integration and acceptance tests have received more recognition thanks to the agile methods [4-7]. Agile methods treat testing as an integral part of their processes. In these methods, no modification or refactoring of code is complete until 100% of unit tests have run successfully, no story is complete until all its acceptance tests have passed successfully, and additions and modifications to the code are integrated into the system on at least a daily basis. Despite this, we still have little insight into the status of these three types of tests. This insight is pivotal for providing feedback for process improvement and for making the overall development process more cost-effective [8,9].

In this paper, we study developers' testing process at *Nomadic Software*. Our goal is to establish its status within the company and identify areas for potential improvements. The study is based on a testing model [10], developed for a traditional heavyweight development context.

The remainder of this paper is as follows. Section 2 presents our research method and the organization studied. Section 3 describes the developers' testing model. Section 4 presents the status of the testing process and Section 5 makes final remarks.

2. Method

This section describes the research method taken in this study. Subsection 2.1 presents the company studied. Subsection 2.2 describes our research steps. Subsection 2.3 presents the questionnaire used in this study. Finally, Subsection 2.4 motivates the sampling method.

2.1. Nomadic Software

We have studied one large Swedish organization. Due to the sensitivity of the results presented herein, the company does not wish to disclose its name. For this reason, we call it *Nomadic Software*.

Nomadic Software is the IT provider of IT services within a larger group of companies, which we call *The Nomad Group*. This group serves the global market with world-leading products, services and solutions ranging from military defense to civil security products. It is operating in more than 100 countries with its headquarters in Sweden.

2.2. Research Steps

Our research consisted of three phases. As shown in **Figure**

1, these are 1) *Prefatory Study*, 2) *Pilot Investigation*, and 3) *Main Investigation*. Each of the phases consisted of three consecutive steps: *Planning*, *Investigation* and *Analysis*. Below, we briefly describe these phases.

In the *Prefatory Study* phase, we acquainted ourselves with Nomadic Software by investigating its processes and the roles involved in them. For this purpose, we studied the organization's internal documentation and made informal interviews with four developers. Our main purpose was to get background information about the company such as its employees, their working patterns and problems, and their opinions about their testing process. This helped us identify a preliminary status of the developers' testing process and generate questions to be used in later research phases.

After having acquainted ourselves with the company and its process, we decided to make a small survey in the *Pilot Investigation* phase. Here, we created a multiple choice questionnaire based on the results achieved in the former phase. The questionnaire was answered by the same four developers who participated in the *Prefatory Study* phase. Our purpose was to test questions and acquire feedback on their variability and expected answers. This helped us determine the appropriateness of the questions and their level of inquiry.

In the *Main Investigation* phase, we first designed a comprehensive questionnaire to be distributed to all the developers within *Nomadic Software*. Our purpose was to achieve a detailed description of the *AS - IS* situation of the company's developers' testing process, its inherent activities, information managed within the process, roles involved and the tools used.

Even if the questionnaire consisted of multiple choice questions, it became very detailed, and of considerable size. *Nomadic Software* estimated that it would take about one hour for each developer to answer it. Having as many as about eighty developers, it would be too expensive. For this reason, we had to cut out many of its questions and/or redesign others. We also had to split parts of the questionnaire into two sub-parts: one studying static testing and the other one studying dynamic testing. All in all, we received answers from fifteen developers, where seven developers answered the dynamic part and twelve developers answered the static part.

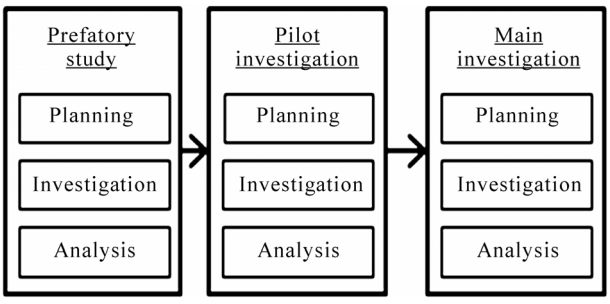

Figure 1. Research steps taken in this study.

2.3. Questionnaire

The questionnaire consists of two main sections: *Background* and *Testing Process*. It is shown in **Figure 2**.

2.3.1. Background Section

The *Background* section inquires about the respondents and the underlying testing conditions within the company studied. It covers the following:

1) *Developers*' *Background*: This part inquires about the developers, their testing experience and current responsibilities. It covers *Questions* 1-3. Our aim is to get to know the developers' background and provide a basis for analyzing the results of this study.

2) *Methods Used*: This part inquires about the development methods defined and used within the company. It covers *Questions* 4-6. Our goal is to find out whether the developers use the methods defined within the company and the reasons behind using or not using them.

3) *Scope and Effort of Testing*: This part inquires about the scope of the developers' testing activities and the effort spent on them. It covers *Questions* 7 and 8. The goal is to find out what testing levels and activities the developers are involved in, what is the effort spent on them, and its distribution on manual and automatic testing.

2.3.2. Testing Process Section

In the Testing Process section, we inquire about the status within the testing phases such as Preparatory, Write Code/ Change Code, Testing, Debugging, Evaluation and Sign-Off (see **Figure 2**).

1) Preparatory Phase: This part inquires about the planning of the developers' work. It includes the following parts:

- Documentation Part: This part inquiring about the documents providing input to the implementation phase. It covers Questions 9-11. Our goal is to find out what documents are studied before the implementation, what measures are taken in cases when they are defective, and elicit examples of the defects.
- Testing Plan: This part inquiring about the developers' test plans. It covers Questions 12 and 13. The goal is to find out what activities are included in the implementation and testing phases and when they are carried out.
- Testing Environment: This part inquiring about the activities the developers conduct to create a testing environment. It covers Question 14. The goal is to find out what activities that the developers create when setting up their own testing environments.

2) Write Code/Change Code Phase: This part inquires about the basic code implementation activities. It covers Question 15. Our aim is to assure that all the respondents write and/or change code, and test it.

Background Section

1) Does your current role entail programming of software?
2) Please state the number of years that you have been working with testing in your current role (at your current employer and in total).
3) What type of activities are you currently involved in?
   ● Development of a completely new system;
   ● Development of a new feature in an existing system;
   ● Defect correction; ● Testing of your own code;
   ● Testing of other developers' code; ● Writing your own test cases; ● Writing other developers' test cases; ● Other, please specify;
4) What test processes/methods do you follow?
   ● Rational Unified Process (RUP); ● Software Development Process (*NomadicRUP*) in all your development; ● Software Development Process, not in your testing; ● Software Test Process (a separate test process linked to *NomadicRUP*; ● Method that you have brought in with you from your former company; ● An old method that has been used earlier at *Nomadic Software*;
   ● Your own method; ● No method at all; ● Other, please specify;
5) If you use use the *Software Development Process (NomadicRUP),* does it contain enough information regarding unit and unit integration test activities?
6) If you follow the *Software Test Process*, do you believe it to be useful for increasing the code quality?
7) What is your weekly effort (in percentage) spent on manual and dynamic testing?
8) Which types of testing are you involved in?
   ● Unit tests; ● Integrations test of you own units;
   ● Continuous integration of components; ● Functional tests;
   ● Security tests; Regression tests; ● Integrity tests;
   ● Other tests, please specify.

Testing Process Section

9) What documents do you study before coding and testing?
   ● Requirements specification; ● Design specification
   ● Program specification; ● Change request; ● Problem report;
   ● Nothing, everything is communicated orally;
   ● Other test, please specify;
10) If some of the documents have inconsistencies, need further clarification or is missing information, do you report that it needs updating?
11) Please, state which documents need updating and list some of the problems identified in those documents.
12) Which of the activities do you include in your testing plan?
   ● Coding; ● Testing; ● Creating stubs and drivers;
   ● Preparing your testing environment; ● Modifying of your regression test cases; ● Others, please specify;
   ● None, you do not create your own testing plan;
13) When exactly do you carry out those activities?
   ● Before coding; ● During coding; After coding;
   ● Never;
14) Do you create your own testing environment? If yes, what exactly do you do?
15) Which activities do you perform in the *Write Code/Change Code* phase?
   ● Write code; ● Change code; ●Link and compile code;
   ● Other, please specify;

Dynamic Testing Part

16) Which types of dynamic testing do you perform?
   ● Black-box tests; ● White-box tests; ● Grey-box tests;
17) Which role(s) does/do write your test cases?
   ● Another developer writes your test cases; ● An integrator writes your test cases; ● A software architect writes your test cases,
   ● Other role writes your test cases, please specify which role it is;
18) Do your document your own test cases?
   ● Always; ● Very often; ● Half of them; ● Rarely;
   ● Never;
19) What are your testing coverage goals?
20) If you have not achieved your testing goals, what do you do?
21) Which method do you use when designing your input data?
   ● Equivalence partitioning; ● Boundary-value analysis;
   ● Cause-effect graphing; ● Error guessing; ● Statement coverage;
   ● Decision coverage; ● Conditions coverage;
   ● Decision/condition coverage; ● Multiple-condition coverage;
22) When comparing the received testing results with the expected ones, what do you do when you find discrepancies?
   ● You make your own notes; ● You hand in a trouble report;
   ● Other, please specify.
23) Which roles do you get in contact with in the following situations?
   ● Who informs you about the functionality that you have to develop? ● Who informs you about that you have to test other developer's code? ● Who informs you about inconsistencies, needs for clarification and missing information? ● Who do you inform that you that you have updated your own test cases? ● Who do your inform that requirements need to be updated? ● Who do you inform about your testing results? ● Who do you inform that your tests have been completed?

Debugging Part

24) If you are debugging code, what do you when you find discrepancies
   ● Study the source code and correct it, if relevant;
   ● Study test cases(s) and correct it/them, if relevant;
   ● Study the requirements specification and take relevant measures, if relevant; ● Study the design specification and take relevant measures, if relevant; ● Others, please specify;
25) What tool(s) are you using for debugging?

Static Testing Part

26) Are you involved in reviews?
   ● You do reviews of your own code; ● You do walkthroughs (review of a peer's code); ● You do formal inspections; ● You do not do any reviews, walkthroughs or inspections;
27) What is the purpose of your reviews?
   ● Code follows; ● Programming guidelines;
   ● Organizational standards; ● Other criteria, please specify;
28) Do you make notes documenting the results of the reviews?
29) Do you report discrepancies encountered during the reviews?

Sign-off Part

30) Do you sign-off your development and test results before delivering your code for integration or system testing?
31) Do you sign-off your development and test result, exactly which artifacts do you sign-off?

Evaluation Part

32) If you look at how developer's testing is being done today, please state what can be improved and motivate your suggestions.
33) If you look at the way the developer's testing is done today, please state what is the best in today's testing procedures and motivate way.

Figure 2. Our questionnaire.

3) Testing Phase: This part inquires about the status within the test execution phase. It includes the following parts:

- Dynamic Test Execution Phase: This part inquires about the status of the dynamic testing process, its timing and results. It covers Questions 16-23. Our aim is to find out whether dynamic testing activities are conducted, how and when they are conducted, what is their outcome and how they are communicated among the roles involved.
- Static Test Execution Phase: This part inquires about the status of the static testing process and its results. It covers Questions 26-29. Here, we wish to find out whether static testing activities are conducted, how and when they are conducted, and what their outcome is.

4) Test debugging phase: This part inquires about the practice of debugging. It covers Questions 24 and 25. Our aim is to find out what is done when defects are discovered and what tools are used for localizing these defects.

5) Sign-Off Phase: This part finds out whether the developers sign off their results. It covers Questions 30 and 31. Our aim is to hear about whether the developers sign-off their code and what artifacts they sign-off.

6) Evaluation Phase: This part inquires about the developers' opinion about the testing process and their suggestions for the process improvement. Our aim is to elicit the developer's recommendations on how to improve their testing process and/or how to preserve its good elements. It covers Questions 32 and 33.

2.4. Sampling Method

Initially, we intended to achieve a full sampling coverage of our respondents. However, as already mentioned, this was considered to be too expensive by the company's management. Hence, only fifteen developers were involved in this study. These individuals belonged to different projects, and they were chosen by their respective project managers. We had no opportunity to influence their selection. For this reason, we have no other choice than to classify the sampling method used in this study as a convenience sampling method.

The convenience sampling method does not allow us to generalize our results with respect to the status of the organization studied. However, it provides an indication of what the status of the developers' testing process looks like.

3. Testing Model

There are not so many process models delineating the developers' testing process. One of the current ones is illustrated in **Figure 3** [10]. It provides a framework for developers' testing phases and their constituent activities. It was developed in a traditional heavyweight context. However, it is even relevant in the context of agile development. By framework, we mean that it covers most of the activities necessary for conducting unit and unit integration tests.

As shown in **Figure 3**, the phases of the developers' testing process are 1) *Preparatory Phase*, 2) *Write Code/ Change Code Phase*, 3) *Testing Phase*, 4) *Debugging Phase*, 5) *Evaluation Phase* and finally, 6) *Sign-off Phase*.

1) Preparatory Phase

The Preparatory Phase consists of two alternative phases. Their choice depends on whether one writes new code or changes an existing one. The changes may concern changes requested by external customers or changes to be conducted due to discovered defects in any of the testing process phases. The activities for these two phases are almost the same. One makes a new low-level design or checks whether or how to make changes to the existing one. One plans for the next testing iteration, that is, one creates/modifies test cases, specifies/checks inputs and expected outputs, and creates stubs and drivers, if necessary. The only difference is that one revises regression test case base in cases when the code is changed.

2) Write Code/Change Code Phase

During the *Write Code/Change Code Phase*, developers write or change their code and compile it.

3) Testing Phase

The *Testing Phase* consists of unit and unit integration testing which, in turn, may be conducted dynamically and statically. Dynamic testing implies testing software through executing it. One starts by checking if the test cases fulfil the given requirements, one creates additional test cases, if needed, links the units and tests them. The test results are then documented and compared to the expected ones. Static testing, on the other hand, implies testing software through reading it. It ranges from informal code reviews conducted by the developers themselves, to reviews conducted by peers, to formal inspections performed by a group of dedicated roles.

4) Debugging Phase

The *Debugging Phase* is conducted in parallel with the other testing phases. Using the testing results, one localizes defects and removes them. It partly overlaps with the activities within problem management process.

5) Evaluation Phase

The *Evaluation Phase* is conducted on two levels. The first level is performed by developers. They evaluate code before sending it for system integration. The second level evaluates the development and testing routines with the purpose of providing feedback for process improvement.

6) Sign-off Phase

Due to the importance of unit and unit integration tests, the developers should sign off that all the components

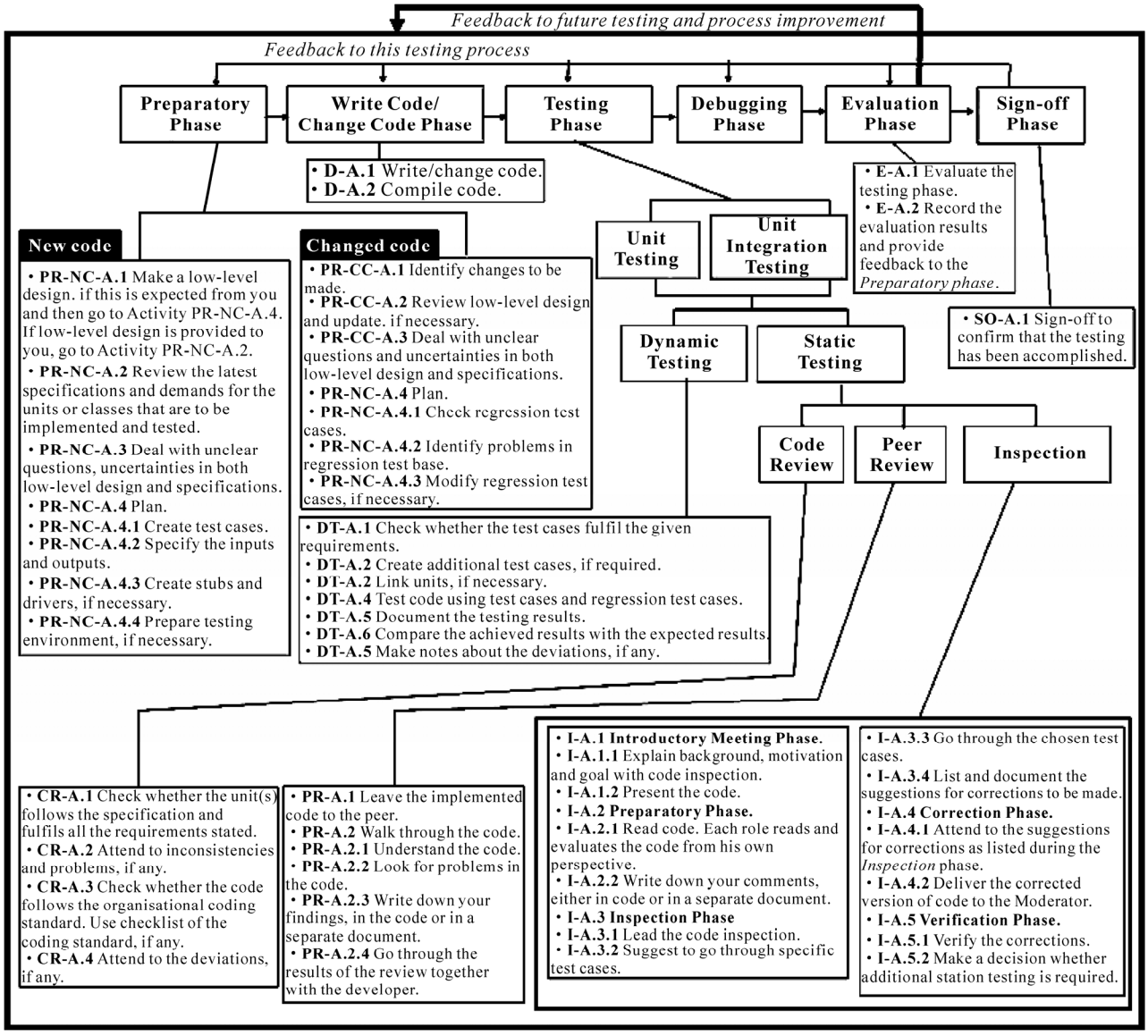

Figure 3. Developers' testing process.

delivered for integration have been successfully tested. We consider the *Sign-off Phase* important because it finalizes the developers' tests. It adds pressure on the developers and hinders them from delivering untested code. It also promotes higher level of accountability among the developers.

The framework does not impose any particular sequence. Developers are free to adapt it to their own context. Usually, before sending their components for integration, they may have to repeat many of its phases or their parts. This is illustrated with a non-bold line in **Figure 3**. In addition, the framework suggests that the developers evaluate the testing process in the *Evaluation* phase and provide feedback for process improvement. This is illustrated with a bold arrow line in **Figure 3**.

4. Status within Nomadic Software

In this section, we present the results of the survey. When reporting on them, we follow the order of the questionnaire as defined in Subsection 2.3.

4.1. Respondents and their Background

All the respondents (100% of response coverage) are involved in programming. As illustrated in **Figure 4**, in average, they have been working with programming and testing for 3.2 ± 3.2 years at *Nomadic Software* and for 7.4 ± 7.1 years in their career lives.

The respondents are involved in various lifecycle phases; 53.3% are involved in developing new systems,

93.3% enhance existing systems with new features and 86.7% attend to software problems. Irrespective of the phase, all the respondents are involved in writing and testing their own code. Out of them, 46.7% write their own test cases and 13.3% write test cases to be used by other developers. Some of them (6.7%) also conduct other unspecified activities.

4.2. Method Used

Nomadic Software has defined and established their own development method. This method is based on RUP [11] and it is called *NomadicRUP*. All the developers are required to follow it either standalone or in combination with the *Software Test Process*, a process that has been defined and established by *Nomadic Software*. Despite this, as shown in **Figure 5**, only 33.3% of the respondents follow it within all their development activities (including testing).

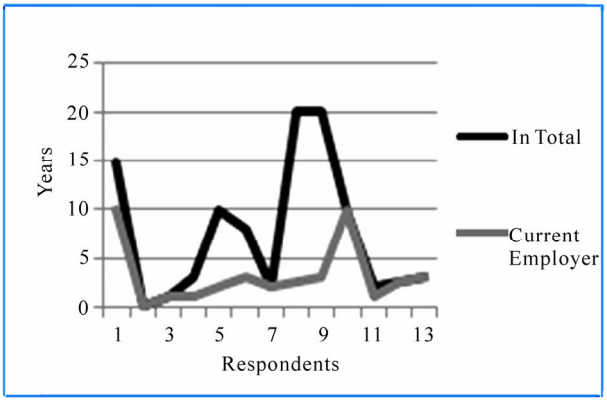

Figure 4. Experience in testing.

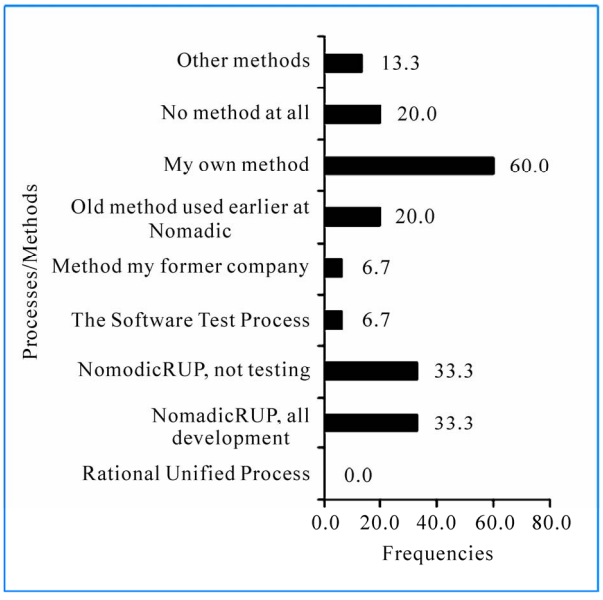

Figure 5. Process/method followed.

Regarding the remaining respondents, 33.3% of them, follow *NomadicRUP* within development but not within testing; 6.7% utilize the *Software Test Process*, 6.7% use a method that they have brought with them from an earlier employer, and 20.0% use an old *Nomadic* method. As many as 60 % use their own method and as many as 20.0% do not use any method at all. Finally, 13.3% of the respondents use methods such as Scrum, XP and ITM Process [12-14].

It is easy to recognize in **Figure 5** that the majority of the respondents follow more than one method. This is proved by calculating the accumulated frequency which is 193.3%.

Our respondents have admitted that they conduct developers' tests in an ad hoc manner. They mainly use common sense when testing their code. However, they claim that they are more disciplined when performing higher-level tests with respect to planning, testing and follow up.

There are many reasons to why *NomadicRUP* is not used by all the developers. Some of: 1) the developers have not even made an effort to get acquainted with the method; hence, they do not use it, 2) the methods are too general and it does not support their specific development needs while the use of Scrum has substantially increased progress and code quality, 3) the developers have gone over to Scrum because they feel that by using *NomadicRUP*, they produce a lot of meaningless and quickly outaging documentation instead of writing code, 4) the developers continue with the *NomadicRUP*'*s* forerunner that was used to develop and that is still used to maintain some of the existing applications, 5) the developers wish to decide by themselves on how to carry out their own testing work.

As shown in **Figure 5**, 66.6% (33.3% + 33.3%) of the respondents follow the *NomadicRUP* method but only 33.3% of them use it for testing purposes. Still, however, 63.7% of them are of the opinion that the method includes sufficient information about developers' testing process.

Regarding the *Software Test Process*, only 6.7% of the respondents follow it (see **Figure 5**). Just as with the *NomadicRUP* method, some of the respondents are of the opinion that even this method includes sufficient information about and guidelines for conducting developers' tests and that it generates better code quality. Some other respondents claim that the very abstract presentation level of the method allows them to state that they follow the method. In reality, however, they use common sense when testing their components.

Irrespective of whether the developers follow the software test process, some of them are of the opinion that is it useful to have a formal testing process on a developers' level. It forces the developers to create test cases on different levels, imposes traceability among them and facilitates future development and change.

However, the main obstacle hindering the developers to do the testing is time. They have little time assigned to do the unit and unit integration tests.

4.3. Scope and Effort of Testing

Testing is mainly done manually at *Nomadic Software*. The respondents had difficulties to estimate the effort spent on the manual and automatic testing. This is because the effort varies from week to week or it depends on the complexity of code. In average, however, as shown in **Figure 6**, the respondents spend $0 \leq 30.9\% \leq 69.2\%$ of their weekly working time (40 hours) on doing manual tests and only $0 \leq 2.4\% \leq 9.1\%$ of their time on doing automatic tests.

Developers conduct various tests. As shown in **Figure 7**, their testing activities range from unit tests (92.9%), through unit integration tests (92.9%), functional tests (71.4%), system regression tests (42.9%), and testing of other developers' integrated components (50.0%). In addition, some of them are involved in tests such as usability tests (28.6%), integrity tests (21.4%), and security tests (21.4%). It is worth mentioning that not all the respondents were familiar with all the test types mentioned in the question.

4.4. Preparatory Phase

Various documents provide basis for starting the coding activity. As shown in **Figure 8**, our respondents mainly use 1) requirement specifications (80.0%), 2) design specifications (73.3%), 3) change requests (86.7%), 4) program specifications (53.3%), and 5) problem reports (60.0%). The use of problem reports supports developers in recreating reported problems and in finding deficiencies in the development and maintenance. However, as many as 13.3% of the respondents use oral communication as a basis for their coding activities. This is because the above-mentioned documents do not always exist. Another reason is the fact that many of the above-mentioned documents are not always of satisfactory quality. Hence, the respondents find it easier to use oral communication as a basis for starting their coding activities.

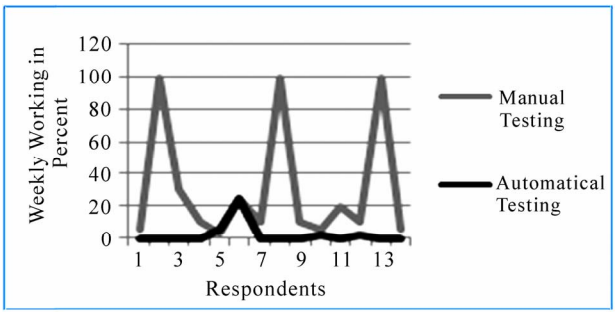

Figure 6. Effort spent on testing.

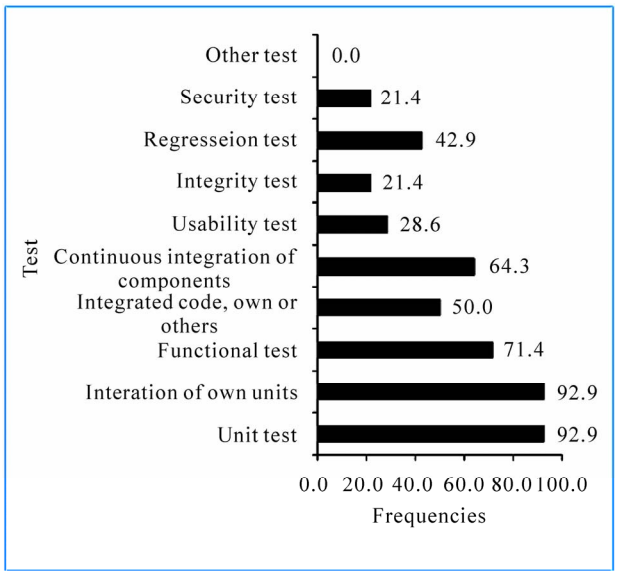

Figure 7. Test conducted.

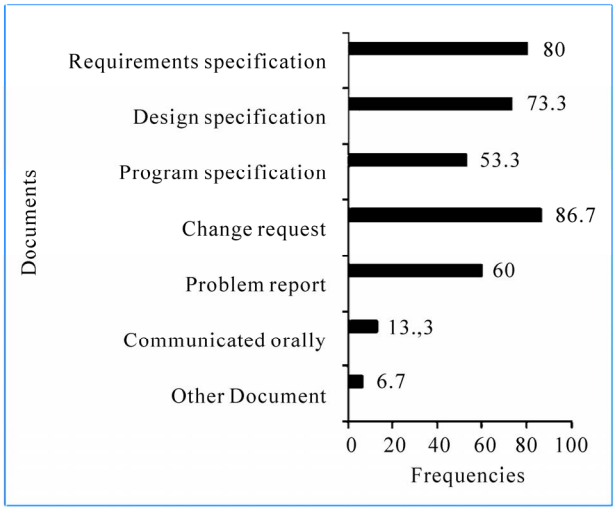

Figure 8. Document.

When studying the above-mentioned documents, the respondents often discover various defects concerning inconsistencies and/or uncertainties. As shown in **Table 1**, the respondents that have answered this question find defects ranging from missing information in design specification to missing or outdated information in various documents. These defects are then reported for corrective measures by 92.9% of the respondents. The reporting is done for the purpose of updating the documents and not for the purpose of providing a basis for improving the testing process. The remaining respondents (7.1%) do not do any reporting at all.

Some of the respondents have not provided any information on what documents they use as a basis for starting their coding and testing activities. They have however provided us with the following opinions: 1) when

Table 1. Defect examples.

Document name	Defect
Requirement specification	A conditions has to few exits Use Cases are on a too high-level Missing business rules
Design specification	Missing information or documents
Change report	Common functionality is not common
Problem report	Outdated information

Table 2. Timing of some testing activities in percentage.

Activity	When			
	Before coding	During coding	After coding	Another time
Stubs and drivers	33.3	66.7	33.3	0.0
New functionality test cases	41.6	41.6	75.0	16.7
Regression test cases	0.0	16.7	83.3	0.0

designing the system I do not add any descriptions about how to conduct unit and integration tests since it is not requested by the organization, 2) I cannot remember a document that does not include inconsistencies and/or uncertainties, and 3) documentation is generally a bad way of communicating information to the implementation process and to keep information about how things work. Therefore, documented tests are a lot better if they are combined with documentation easily extractable from code.

The respondents were asked to list the activities that they included in their testing plans. Only 71.4% of the respondents plan their implementation and testing. In their plans, they include 1) coding (90.0% of the respondents), 2) testing (100%), 3) preparation of their own testing environments (80.0%), and 4) modification of regression test cases (40.0%). Very few of the respondents (20.0%) include creation of stubs and drivers in their testing plans. These plans, however, are made on an informal basis. This is because the organization does not promote planning of and documenting tests.

Regarding the activities included in the testing plan, we inquired about the point in time when they were conducted. Our aim was to find out whether they were conducted 1) before coding, 2) during coding, 3) after coding, or 4) never. As shown in **Table 2**, the timing of these activities varies in the following:

1) Stubs and drivers are created before and during coding.

2) Regression test cases are modified during and after coding. However, the greater majority of the respondents (83.3%) modify them after they have finished coding.

3) New functionality test cases are written throughout the whole implementation process. The majority of the respondents (75%), however, create them after coding.

We also inquired whether the respondents created their own testing environments and exactly what they did when doing it. Eighty percent of the respondents do create their own testing environments. When doing it they (1) test project code and run functional testing of their own components, (2) change test data by making a copy of production data, (3) use remote automatic tests whenever they are checking something in, (4) create a number of settings to point out the resources required to run and execute a build. In addition, the respondents have commented that they have three environments: development, test, and production. However, they only use the development environment when conducting developer's tests (unit tests).

4.5. Write Code/Change Code Phase

All the respondents (100%) write new code and change an existing code. However, 61.6% of the respondents have to compile their code manually. The remaining respondents get it automatically done via tools which both check syntax and compile the code.

4.6. Dynamic Testing

The respondents were requested to list the dynamic testing practices they used. The majority of them (85.7%) conduct black-box and white box tests. Although grey-box testing is not promoted at *Nomadic Software*, 42.9% of the respondents have answered that they conduct grey-box tests as well.

We inquired whether the respondents wrote test cases by themselves or whether they got them written by some other role. We also inquired if they documented their own test cases. Our results show that all the respondents claim that they write their own test cases, but on some occasions, 14.3% of them use test cases written by other developers. No other role than a developer role is involved in writing test cases for our respondents.

We inquired whether the respondents documented their own test cases. Our results show that 42.9% of the respondents always document their test cases, 28.6% do it very often, 14.3% do it rarely, and 14.3% never do it. Some of the respondents have pointed out that one mainly puts effort into documenting the integration test cases instead. Other respondents have mentioned that documentation is only in JavaDoc but that they can generate a report on all tests when they run them.

Developers' tests are the most efficient tests to conduct. Because the cost of coverage is low, one should strive to set a testing coverage goal as high as possible.

We inquired about the developers' coverage goals. Our results are illustrated in **Figure 9**. As shown there, 1) 33.3% of the respondents test all main parts of code, 2) 50% test all code, 3) 16.7% test 60%-80% of all code 4) 16,7% test all features, and 5) 16.7% test all the architectural decisions. These results do not specify testing coverage for any specific testing technique. This is because the coverage goals are not determined by the organization but by the developers themselves. However, as **Figure 10** shows, the white-box testing techniques used by the respondents are 1) multiple-conditions coverage, 2) decision/conditions coverage, 3) condition coverage, 4) decision coverage, and 5) statement coverage.

Regarding the test cases involving input data, most of the respondents (80% of them) use the boundary analysis method, and 60% use error guessing. Some of them also use equivalence partitioning (40%) and cause-effect graphing technique (20%). In cases when the coverage goals are not achieved, the respondents take measures such as 1) discuss cost and revenue of further testing, 2) ask the project manager for further measures, 3) decide by themselves what to do next, or 4) they just checked in code to the repository.

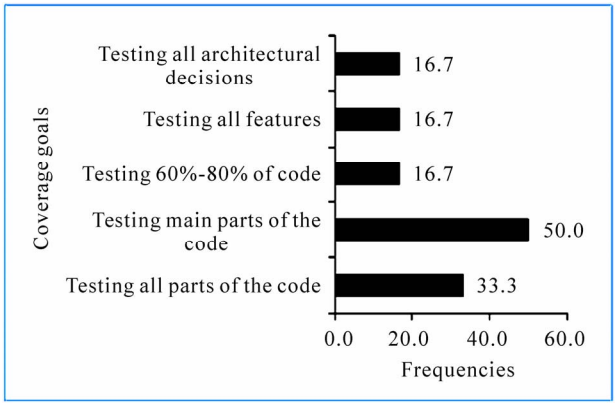

Figure 9. Test coverage goals.

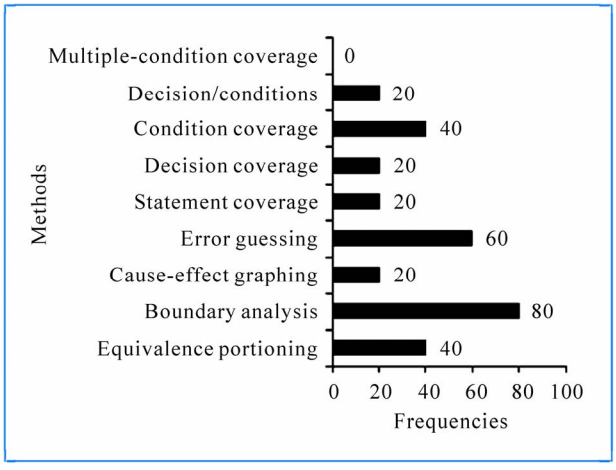

Figure 10. Data input methods used.

Test results ought to be documented. For this reason, we inquired whether the respondents recorded their testing outcome and how they did it. Our results show that 57.1% of the respondents make their own informal notes about the discrepancies between the expected and achieved results and 28.6% hand in trouble reports, if necessary. Some of the respondents (42.9%) not only make notes or hand in trouble reports but also correct the code by themselves. Finally, some of the respondents just fix code without making either formal or informal notes.

Developers come in contact with various roles in different situations. These are:

1) *System Analysts* and *System Architects* to discuss new functionality to be developed, suggestions for their updates and reports on inconsistencies in them, if any.

2) *Business System Manager* and *End User* to discuss maintenance tasks and inconsistencies in them, if any.

3) *Test Managers* requiring that the respondents test other developers' tests. The respondents may also inform the *Test Managers* about the completion of their tests and their testing results.

4.7. Test debugging Phase

We inquired about how the developers tracked defects in the *Debugging* phase and what tool support they used. Our results show that all the respondents debug their code, if needed. If they find defects, 100% of them correct them in source code and requirements, and 85.7% correct them in design specifications and test cases. The tools used during the *Debugging* phase are, for instance, *Visual Studio*, *IntelliJ*, *JProfiler* and *Jboss*.

4.8. Static Testing

Given a set of static testing practices, the respondents were requested to identify the ones they used. They had a choice of 1) own reviews implying that they checked their own code, 2) walkthroughs of peer code, and 3) formal inspections. Our results show that 100% of the respondents review their own code, 9.1% do walkthroughs of peer code, and 18.1% are involved in inspections. The inspections, however, are very seldom performed.

We inquired about the purpose of the reviewing activities. Irrespective of how the developers review their code (own review or walkthroughs), at least 90.1% of the respondents review it for the consistency with the requirements. When conducting own reviews, 63.6% of the respondents also review for organizational standards, and 9.1% review for other criteria such as, for instance, international standards. In the context of walkthroughs, 18.2% of the respondents review for organizational standards only. In situations when the respondents con-

duct inspections, they only review for consistency with requirements.

In static testing, it is imperative to document the testing results. Hence, we inquired whether the respondents recorded them. Our results show that very few respondents, only 18.2% of all of them, document the results of their own code reviews and no one documents walkthrough and inspection results.

We also inquired how the respondents documented the discrepancies discovered during static testing. As illustrated in **Table 3**, the results span between 54.5% of the respondents making own notes to 9.1% of the respondents handing in trouble reports. The remaining discrepancies are communicated on an oral basis.

4.9. Sign off

We asked the respondents whether they finalized their implementation and testing activities in a formal or informal way, for instance, by signing off their code. Only 9.1% of the respondents sign-off their work after they have completed their tests. The artifact that is used for signing-off is mainly a version management tool complemented by an informal hand-shake among the developers, testers and managers.

4.10. Evaluation

We also inquired about the best of the today's testing process. According to the respondents, the best parts of the process are 1) the ability to conduct test review, 2) freedom to use, for instance, Scrum/XP instead of, for instance, *NomadicRUP*, 3) the ability to import production data to be used as test data, 4) the ability to test your own code, 5) automatic test framework, 6) the opportunity to start testing early in the development cycle. Their motivations are 1) system development using Scrum/XP generates less defects, 2) test data can always be up to date since it is possible to copy production data, 3) the framework automatically conducts regression tests, 4) by placing testing early in the development cycle, focus is set on the actual problems and assures that test cases are written, and finally, 5) the transfer of documents is substantially reduced.

Table 3. Recording Testing Results

Methods	I make own notes	I hand in trouble reports	Other, please specify	I do not document
Own Review	54.5	18.2	9.1	18.2
Walkthroughs	9.1	0	9.1	64.7
Inspections	27.2	9.1	9.1	45.5

5. Final Remarks

In this paper, we have studied developers' testing process at *Nomadic Software.* Our goal is to establish its status within the company and identify areas for potential improvements. The study is based on a traditional testing model elicited in [10]. The respondents involved in this study are developers with solid programming background and experience.

Our results show that the developers' testing process is not uniformly performed within *Nomadic Software*. Right now, the company suffers from the following problems:

1) *Lack of control over the methods used*: Even if *Nomadic Software* has put effort into defining and establishing a development and testing process, the majority of the developers still use other methods and they conduct their tests in an ad hoc manner. Irrespective of the reasons behind, *Nomadic Software* did not have insight into what methods were used within the company before this study. Neither did it have control over the status of the developers' testing process. Regarding the developers, some of them are hardly acquainted with the company's testing method.

2) *The organization does not assign enough time for conducting developers' tests*. This leads to the fact that developers' tests get neglected. Developers are too much in a hurry to deliver code for integration and system tests.

3) *Testing coverage goals are not clearly stated by the organizations studied*. Neither are they determined for any specific testing technique. This implies that each developer sets his own goals. This, in turn, may lead to strongly varying code quality as delivered by various developers.

4) *Important requirements and defects in requirements specifications are communicated orally*: Quite a big portion of requirements and problems are communicated orally. These requirements and problems do not get documented even after being implemented. This is a severe problem that may substantially degrade the system maintainability and contribute to quick software ageing and lack of control over the development and maintenance process [15].

5) *Not all test cases get documented*: This implies that the company cannot determine whether the developers' testing has been sufficiently performed. This also implies that regression testing on the developers' level practically does not exist.

6) *Static testing is not practiced enough*: Static testing is performed on an informal basis. At its most, developers review their own code and sometimes their peers' code. Formal inspections of critical code parts are conducted very seldom.

7) *Lack of testing guidelines*: Lack of testing guide-

lines makes developers decide by themselves on how to conduct their testing activities. This, in turn, leads to the non-uniformity of the testing process execution.

8) *Insufficient education within testing*: The employees at *Nomadic Software* get a very short education on development method, where testing is one of its parts. Hence, they have not acquired sufficient knowledge. This is clearly evident from the fact that the respondents are not acquainted with some basic testing terms such as integrity tests or they use the terms differently. A similar phenomenon has been observed in our former study in [16].

9) *Lack of testing strategy*: *Nomadic Software* lacks a strategy aiding them in defining how to test in a cost-effective and qualitative manner and designating test types to be part of the testing process.

Due to the sampling method used in this study, we cannot generalize the results presented herein. However, we may still claim that our results strongly indicate that just as *Nomadic Software*, many software companies are in great need to revise their developers' testing process, put it in the context of its overall testing process and make effort into improving it.

When studying the developer's testing process at *Nomadic Software*, we have identified several problem areas related to the education of developers and the management and execution of the testing process. Specific pains that we have observed are lack of control over the testing methods used, lack of testing strategies and lack of directives of what is expected from the developers. To attend to these problem areas is not an easy task. It requires many different measures ranging from creating appropriate overall testing strategies in which developer's testing strategy is clearly identified and specified, defining testing processes in which developers' tests play an essential role, and monitoring that they are followed by the developers. To realize them can be a long and complex process. However, as an initial step towards improving the developers' testing process, we suggest the software community create guidelines providing instructions and recommendations specifying what and how developers' tests should be done and what sort of actions should be taken in particular testing circumstances.

6. References

[1] J. W. Cangussu, R. A. DeCarlo and A. P. Mathur, "A Formel Model of the Software Test Process," *IEEE Transactions on Software Engineering*, Vol. 28, No. 8, 2002, pp. 782-796.

[2] L. Groves, R. Nickson, G. Reeves, S. Revves and M. Utting, "A Survey of Software Pratices in the New Zeeland Software Industry," *Proceedings of Australian Software Engineering Conference*, Queensland, 28-29 April 2000, pp. 189-201.

[3] S. P. Ng, T. Murnane, K. Reed, D. Grant and T. Y. Chen, "A Preliminary Survey on Software Practices in Australia," *Proceedings of Australian Software Engineering Conference*, Melbourne, 13-16 April 2004, pp. 116-125.

[4] "Agile Software Development," 2009. http://en.wikipedia.org/wiki/Agile_software_development

[5] H. Gallis, E. Arisholm and T. Dyka, "An Initial Framwork for Research on Pair Programming," *Proceedings of ISESE International Symposium on Empirical Software Engineering*, Rome, 30 September-1 October 2003, pp. 132-142.

[6] E. M. Guerra and C. T. Fernandes, "Refactoring Test Code Safely," *Proceedings of ICSEA International Conference on Software Engineering Advances*, Cap Esterel, 25-31 August 2007, p. 44.

[7] P. J. Schroeder and D. Rothe, "Teaching Unit Testing using Test-Driven Development," 2005. http://www.testingeducation.org/conference/wtst4/pjs_wtst4.pdf

[8] S. Koroorian and M. Kajko-Matsson, "A Tale of Two Daily Build Projects," *Proceedings of International Conference on Software Engineering Advances*, Porto, 20-25 September 2009, pp. 245-251.

[9] G. J. Meyers, T. Badgett, T. M. Thomas and C. Snadler, "The Art of Software Testing," 2nd Edition, John Wiley & Sons, Inc., Hoboken, 2004.

[10] M. Kajko-Mattsson and T. Björnsson, "Outlining Developer's Testing Mode," *Proceedings of EUROMICRO Conference on Software Engineering and Advanced Applications*, Lübeck, 27-31 August 2007, pp. 263-270.

[11] B. Henderson-Sellers, G. Collins and I. Graham, "UML-Compatible Process," *Proceedings of* 34*th Annual Hawaii International Conference on System Sciences*, Maui, Vol. 3, 3-6 January 2001, p. 3050.

[12] "ITM Process (IT-Product Maintenance Process)," Internal Documentation at Nomadic Software, 2009.

[13] R. Juric, "Extreme Programming and its Development Practices," *Proceedings of* 22*nd ITI International Conference Information Technology Interfaces*, Pula, 13-16 June 2000, pp. 97-104.

[14] L. Rising and N. S. Janoff, "The Scrum Development Process for Small Teams," 2000. http://members.cox.net/risingl1/Articles/IEEEScrum.pdf

[15] M. Kajko-Matsson, "Corrective Maintenance Maturity Model: Problem Management," Ph.D. Dissertation, Stockholm University and Royal Institute of Technology, Stockholm, 2001.

[16] M. Kajko-Mattsson, "Common Concept Apparatus within Corrective Software Maintenance," *Proceedings of International Conference on Software Maintenance*, Los Alamitos, 30 August-3 September 1999, pp. 287-297.

