Available online at www.sciencedirect.com
SCIENCE DIRECT
The Journal of Systems and Software 79 (2006) 1001–1014
The Journal of Systems and Software
www.elsevier.com/locate/jss

Results from introducing component-level test automation and Test-Driven Development

Lars-Ola Damm [a,b,*], Lars Lundberg [b]

[a] Ericsson AB, Ölandsgatan 1, Box 518, SE-371 23, Karlskrona, Sweden
[b] School of Engineering, Blekinge Institute of Technology, Box 520, SE-372 25 Ronneby, Sweden

Received 10 September 2004; received in revised form 24 October 2005; accepted 24 October 2005
Available online 5 December 2005

Abstract

For many software development organizations it is of crucial importance to reduce development costs while still maintaining high product quality. Since testing commonly constitutes a significant part of the development time, one way to increase efficiency is to find more faults early when they are cheaper to pinpoint and remove. This paper presents empirical results from introducing a concept for early fault detection. That is, an alternative approach to Test-Driven Development which was applied on a component level instead of on a class/method level. The selected method for evaluating the result of introducing the concept was based on an existing method for fault-based process assessment and was proven practically useful for evaluating fault reducing improvements. The evaluation was made on two industrial projects and on different features within a project that only implemented the concept partly. The evaluation result demonstrated improvements regarding decreased fault rates and Return On Investment (ROI), e.g. the total project cost became about 5–6% less already in the first two studied projects.
© 2005 Elsevier Inc. All rights reserved.

Keywords: Component testing; Test-Driven Development; Fault metrics; Software process improvement

1. Introduction

Avoidable rework is commonly a large part of a development project, i.e. 20–80% depending on the maturity of the organization (Shull et al., 2002). Having many faults left to correct late in a project leads to higher verification costs. That is, several studies demonstrate that faults are cheaper to find and remove in early stages of projects (Boehm, 1981; Shull et al., 2002). Further, having fewer faults leads to improved delivery precision since software processes become more reliable when most faults are removed early (Tanaka et al., 1995).

This paper presents the result of the implementation of a concept for achieving early fault detection in two commercial development projects at a software development department at Ericsson AB. In order to achieve this, the department wanted to develop a framework for automated component testing. A previous publication proposed such a framework based on component-level test automation where the test cases are written before the code. That is, an alternative approach to Test-Driven Development (TDD) (Beck, 2003). The purpose of this paper is to provide results from implementing the proposed concept in two commercial projects. Thereby, the primary research question is as follows:

What are the costs and benefits of introducing a framework for component-level test automation and Test-Driven Development (TDD) in an industrial setting?

When implementing such changes in an industrial setting, it is not as easy to evaluate the result as for controlled research experiments. That is, in commercial projects, it is

* Corresponding author. Address: School of Engineering, Blekinge Institute of Technology, Box 520, SE-372 25 Ronneby, Sweden. Tel.: +46 455 395561; fax: +46 455 815 10.
E-mail addresses: lars-ola.damm@ericsson.com, lars-ola.damm@bth.se (L.-O. Damm), lars.lundberg@bth.se (L. Lundberg).

0164-1212/$ - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2005.10.015

likely that several other variables than those that were part of the planned change are affected, e.g. new project members and increased organizational maturity. Further, many Software Process Improvement (SPI) frameworks such as CMM and ISO 9000 advocate implementing many improvements at the same time (Conradi and Fuggetta, 2002). Therefore, it is for example hard to relate cost and lead-time changes to certain improvement actions.

However, since the primary output of testing is found faults, a possible approach to such an evaluation is to look at differences in fault distributions. However, in order for such an approach to be useful, the following two criteria should be met:

1. The approach should be able to quantify the obtained benefits of the improvement; just classifying faults into categories is not enough. Preferably, the Return On Investment (ROI) should be possible to obtain.
2. Since several differences between projects might affect the fault distributions, the approach should be able to relate differences in fault distributions to a certain change.

Classification of faults from their causes, e.g. root cause analysis is a very common approach to fault classification (Leszak et al., 2000). However, neither root cause analysis, nor other fault classification approaches such as ODC triggers (Chillarege et al., 1992) are applicable in this context since they are not feasible for cost benefit analysis. Further, there are techniques such as 'Six Sigma' (Biehl, 2004) that evaluate quality improvements by measuring differences in fault distributions in relation to product size or fault detection phase. However, these techniques cannot relate differences in fault distributions to a certain change, and are as further described in Section 2.2 not suitable for evaluating the efficiency of the verification process.

Another possibility is to measure the number of faults that *should* have been found in an earlier phase, i.e. 'faults-slip-through' (FST) (Damm et al., 2004). Although this method was originally intended for process assessment, it could also be used for quantifying the benefits of an improvement that should result in finding more faults early. That is, the method also measures the Avoidable Fault Cost (AFC) in the assessed projects by multiplying the number of FST with the average fault costs in different test phases (Damm et al., 2004). The method can also compare fault costs in relation to specific phases. Therefore, it should also be possible to determine if the differences in fault costs matches the phases where an improvement should have had an effect (Damm et al., 2004). Thus, the method satisfies the criteria for being able to evaluate the primary research question. Consequently, the second research question is:

How can costs of FST be used for quantifying the benefits of software test process improvement?

This paper is organized as follows. After an overview of related work in Section 2, the proposed method is described in Section 3. Then, the practical applicability of the method is demonstrated when evaluating the primary research question in Section 4. After that, a discussion regarding the value, validity and applicability of the results is provided. Section 6 concludes the work followed by two appendices that describe underlying calculations for the obtained results.

2. Related work

Before describing the method and case study results, this section maps the implemented concept and the selected result evaluation method to related research work. Section 2.1 describes work that is related to the first research question stated in Section 1, and Section 2.2 addresses the area of the second research question.

2.1. Early fault detection and Test-Driven Development

Unit testing and inspections are common techniques for detecting faults early. Several widely used techniques for testing units based on their internal structure exist, e.g. path testing, random testing and partition testing (Beizer, 1990). Some unit test techniques implement assertions directly in the product code. Test-Driven Development (TDD) is one technique that has such an approach (Beck, 2003). Tools such as JUnit or CPPUnit are often used together with TDD (Beck, 2003). The main difference between TDD and a typical test process is that in TDD, the developers write the tests before the code. A result of this is that the test cases drive the design of the product since it is the test cases that decide what is required of each unit (Beck, 2003). Therefore, TDD is not really a test technique (Beck, 2003; Cockburn, 2002); it should be considered as a design technique. In short, a developer that uses traditional TDD works in the following way (Beck, 2003):

1. Write the test case.
2. Execute the test case and verify that it fails as expected.
3. Implement code that makes the test case pass.
4. Refactor the code if necessary.

TDD has been successfully used in several cases within agile development methods such as eXtreme Programming (XP) (Beck, 2003; Rasmusson, 2004). A few experiments and case studies on the effects of such traditional TDD have been performed. Most studies have focused on effects on quality and productivity. Although the studies have been performed either in experimental settings or as isolated small-scale case studies, some trends have been observed. The most apparent effect that TDD seems to bring is that the concept increases the amount of unit testing performed (Erdogmus and Morisio, 2005; George and Williams, 2004). Thus, it is not surprising that at least one study has shown that TDD tends to increase the quality of the delivered code (Maximilien and Williams, 2003). However, the productivity effect of TDD seems to be uncertain.

Download English Version:

https://daneshyari.com/en/article/10348893

Download Persian Version:

https://daneshyari.com/article/10348893

Daneshyari.com