Research on the Phenomenon of Software Drift in Software Processes

Yutao Ma, Jianxun Chen, and Jianghua Wu
State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072, China
College of Computer Sci. and Tech., Wuhan University of Sci. and Tech., Wuhan 430081, China
College of Computer Sci. and Tech., Harbin Engineering University, Harbin 150001, China
mmytt_2000@etang.com, cjxwh@21cn.com, wujianghua658@sohu.com

Abstract

Software drift is a common phenomenon existing in software processes. Negative drift makes function, performance and even overall quality of a software product deviate from customers’ expectations, which will bring about the loss to users and software organizations in succession. In this paper, software drift and its relevant concepts are defined initially. Then, an evolutional model of negative drift is presented, and detrimental effects of the model are also analyzed. How to analyze, measure, and control negative drift is an essential problem to be solved in software evolution. Eventually, we put forward a research framework to guide future work to measure and control negative drift for improving software processes.

1. Introduction

Nowadays, theories and methods of software process have been widely applied in software manufacture and management. Software organizations can improve their capabilities of software processes and reduce risks of software development through effectively defining, measuring and controlling software processes. A mature software process can ensure sound quality of software manufacture (functional correctness, reliable estimation of cost and schedule, etc.), but whether it can or not guarantee the customer’s expectations (or software specifications) still holds indefinite. Although many software organizations (or companies) in U.S. have achieved high level of CMM (Capability Maturity Model) [1]/CMMI (Capability Maturity Model Integration) [2], software bugs are so prevalent and so detrimental that they cost the U.S. economy an estimated $59.5 billion annually [3].

This shows the fact that the problem of dynamic consistency between software requirements and software implementation hasn’t been effectively solved yet, especially the fact that there exists a shortage of methods (or strategies) to make dynamic measurement and control on defects in artifacts of activities in early development phases. Thus, it may cause function, performance and even overall quality of a software product (or its components) to deviate from the customer’s expectations (or software specifications) in development processes, which is so-called software drift. The higher degree of negative drift is, the greater harmful effects on function, performance and even overall quality of a software product will be. Then, the more loss software bugs, or software errors coming from it will bring to users and organizations.

This paper is organized as follows. Section 2 defines software drift and relevant concepts. Section 3 presents an evolutional model of negative drift, and its detrimental effects are also being analyzed. Section 4 puts forward a research framework to analyze and measure software drift. In the end, section 5 comes to a summary and a conclusion for the paper.

2. Software drift

Software drift is a common phenomenon in software processes, and it is the process that causes function, performance and even overall quality of a software artifact (or product) in software processes to deviate from the customer’s expectations (or software specifications). We call the process producing an artifact (or product) that can’t meet anticipant function, performance and overall quality as negative drift (ND); contrarily, the process producing an artifact (or product) that has better function, performance and even overall quality than the customer’s expectations (or software specifications) can be defined as positive drift. In the processes of both positive and negative drift, we call
the degree of drift from original expectations (or specifications) to final artifact (or product) on overall quality (or single quality attribute) as absolute degree; similarly, we call the degree of drift between an artifact (or product) and its staged goal (or specifications) on overall quality (or single quality attribute) as relative degree.

On the one hand, because it’s difficult to have software artifact (or product) met its expectations (or specifications) completely, we can accept ND within the permitted range. But if it exceeds the permitted range and because of the accumulation of relative degrees, absolute degree will be greater, and the magnification effect of software errors coming from it will be more obvious. Therefore, we consider that the magnification effect of software errors is the real instance of ND. On the other hand, positive drift that comes from creative work may increase complexity and cost of a software project, and there is little research on how to define and estimate positive drift at present. So, the research is favorable to balance the relations among complexity, quality, cost and benefits to manage development activities, members and resources in effect.

3. Evolutional model of negative drift

Section 2 introduces the important static characteristics (property and degree) of software drift. However, from the point of view of dynamic process, software drift has its own evolutionary rules, and it may get evolved continuously during development processes and be represented with a certain form in the end.

3.1. Evolutionary process

Traditional Reliability Researches [4,5] indicate that software failures and software errors resulting in disastrous results come from software defects in early phases of development processes. There are many kinds of software defects, and their influences on function, performance and overall quality of a software artifact (or product) are different. However, we believe that, in the essence, a software defect comes into being when absolute degree of primary negative drift (PND, the result of an initial ND process) exceeds the permitted range at a certain time, and it is relevant to the degree of ND and the permitted range of estimation. Furthermore, if a software defect comes out, it will exist in the whole software lifecycle unless being eliminated.
The above process is a typical evolutionary process, and it exists in the whole software lifecycle. The key of the evolutionary process is the transformation from a slight PND to a software defect, and the involved evolitional mechanisms and control strategies are the core of further researches.

3.2. Detrimental effects

In the typical evolutionary process, PND (may exist in requirement and analysis phase), software defect (may exist in requirement, analysis, and design phase), software error (may be found in design and implementation phase), software failure (may be exposed in test and deployment phase) and evolitional mechanisms among them make up of the evolitional model of ND in software lifecycle. The model has the notable detrimental effects as the following:

*Domingo effect*. If a serious PND cannot be found and controlled in time in early development phases, there will be chain reaction (the probability of event to happen is great) in development processes, and a software failure as the ultimate form of evolutionary process perhaps results in serious consequences.

*Butterfly effect*. Although a slight PND is often thought to be trivial in early phase of software lifecycle, if we don’t make effective analysis and control, it may cause serious unavoidable results in the later development phase, which brings great loss to users and development organizations.

*Ripple effect*. The longer a serious PND exists in software artifact (or product), the wider harms may ripple (may cause concurrent errors), and the greater the extent of harms is likely to be. Thus, if we control the transformation from a PND to a software defect earlier, harms of ND will be reduced greatly.

3.3. A case in real development process

This section demonstrates the evolution of PND through developing a simple ATM system (use cases are described in [8]). Operation processes of the system are as follows: client inputs passwords after inserting his card; when the identification has been validated, client can follow the hints of operating system to query balance, withdraw and transfer money. *Handle exceptions* use case is responsible for handling exceptions in abnormal situations during withdrawing and transferring money. We define abnormal situations as events with small probability such as network congestion, where the system should cancel the transaction and ensure client not to suffer the loss. However, clients generally demand finishing transactions in short time (an ambiguous requirement results in a slight PND), if the time for communication is longer than response time, the system does consider that the operation has finished, and then the amount of money in client’s account will be decreased, but in fact the transaction fails. So, from the viewpoint of test, we trace the evolution of PND in a typical iteration workflow (shown as Figure 4) to show its evolutionary process described in section 3.1.

![Figure 4. Evolutionary process of a PND in real case from the viewpoint of software test](image)

4. Research framework for future work

On the one hand, there are close relations among PND, software defect, software error and software failure, so they make up of an integrated and well-organized unit characterizing ND from the angle of dynamic process, which remarkably contrasts to the traditional Reliability Research and Software Quality Engineering [9]. Thus, the further researches of ND should include mechanisms of PND’s generation, the taxonomy of different factors resulting in ND, evolutionary rules, methods of estimation on ND, and control strategies. On the other hand, positive drift is triggered by requirement changes, creative work, etc., it has complex effects on quality attributes of software products and will increase complexity and cost of software development, so it needs more attentions.
Hence, they compose a relatively integrated research architecture based on Software Process and Software Quality Engineering.

Figure 5. The research framework for future work

5. Conclusions

Software drift is a common phenomenon in software development processes. Because of ND, the sound quality of software manufacture may not ensure the customer’s expectations (or specifications). Primarily, we define software drift and relevant concepts. Then, a typical evolutionary process from a PND to a software failure is presented. Eventually, we propose a research framework for future work. The goal of the research is to solve the problem of dynamic consistency between software requirements and software implementation (viz. to analyze influencing factors and evolitional mechanisms of software drift in software processes, to investigate methods of its dynamic evaluation, and to search for strategies and techniques to control negative drift within the permitted range) and to form a suit of new theories and methods, which provides basic academic and technical guarantee for enhancing quality of software products, reducing costs on maintenance and modification, and enlarging benefits of software organizations.

Acknowledgement

This work is supported by National defense pre-research task of “Tenth Five-Year-Plan” of China under grant NO.41306050103, NO.41306070201 and Key Scientific Research Project of Department of Education of Hubei Province of China under grant NO.Z200511005. We gratefully acknowledge invaluable comments from anonymous reviewers.

References