

A Control Approach for Agile Processes

João W. Cangussu
Department of Computer Science
University of Texas at Dallas
cangussu@utdallas.edu

Richard M. Karcich
Pillar Data Systems
Longmont, CO 80503
rkarcich@pillardata.com

Abstract

Agile processes can provide early defect information on all the stages of the development process. The use of a feedback control model to regulate the process based on this information is proposed here. The model was originally designed to regulate the testing process and has proven to be effective. Additionally, an ODC filter is proposed to properly interconnect the models associated with each development phase. The approach appears to be a reasonable solution for the control and prediction of Agile processes.

1 Introduction

Agile processes have been proposed to expedite software production by decreasing documentation overhead and focusing on customer satisfaction through a continuous delivery of the system [1]. One of the main characteristics of Agile Processes is the progressive and constant testing/verification/validation of the produced artifacts [1]. Also, the process is very flexible and should be able to be adjusted when nonconformances are detected. The customer participation in these tasks is an important guidance toward the development of the right product but does not help on adjusting the process.

Here we propose the use of a feedback model to control the process. The model was originally designed to control the testing phase and has been proven effective [2]. The use of the same model to control software development processes using an Agile approach is based on the assumption that inspections on requirements and design are in place and early code is produced. The defects captured during inspections as well as during testing can then be used to feed the model. However, not all defects are related to the phase where they have been detected and a filter is needed to direct these defects to the proper phase. Orthogonal Defect Classification (ODC) [3] is used to create such filter to interconnect the models associated with different phases of the development process.

The remainder of this paper is organized as follows. A brief description of Agile Processes and ODC are presented in Sections 2 and 3 respectively. The foundations for the creation of an ODC Demultiplexer are delineated in Section 4 and some experimental results are presented in Section 5. Section 6 presents the concluding remarks.

2 Agile Processes

An integral part of an ‘Agile-like’ development effort is a process for detecting and removing design defects in the phase in which they are injected by implementing static test techniques. This process emphasizes the static testing of specifications before writing code from those specifications. This process helps the tester prepare to validate a product’s behavior against its functional specification and determine if a product’s architecture and behavior support later dynamic test strategies and tactics [1].

Experience has shown that implementing ‘Agile-like’ development efforts can result in a decrease in the number of failures found during subsequent dynamic testing and a corresponding decrease in debugging, repair and retest time. We can divide static testing broadly into two parts: 1) the review of the architectural specifications for the support of later test strategies and tactics and 2) the static testing and updating of the functional specification. The steps of an ‘Agile-like’ development effort with developers and testers participating are: (i) Analyze the functional and architectural specifications for support of test strategies and tactics; (ii) Identify application rules; (iii) Write assertions; (iv) Identify objects and states; (v) Review assertions; (vi) Validate and revise assertions and specifications; (vii) Design tests; and (viii) Validate test designs. The bottom line with this ‘Agile-like’ approach is that it complements the investment in the specification of the operational and functional systems and in the low-level design by emphasizing the detection and removal of software faults close to their point of introduction.

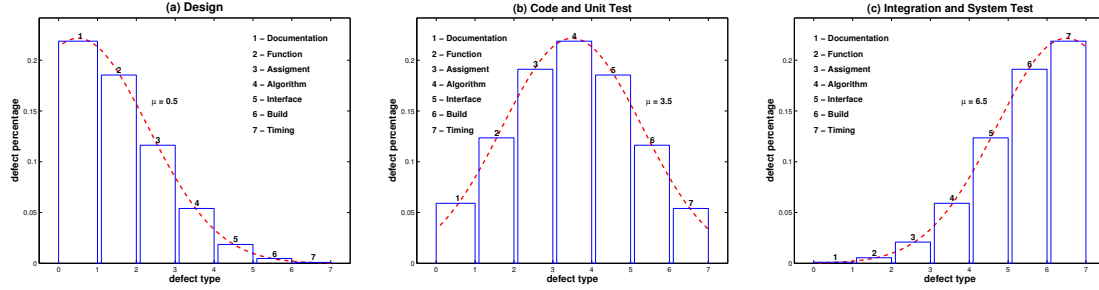


Figure 1. Expected defect type distribution associated with phases of the SDP [3].

3 Orthogonal Defect Classification

Orthogonal Defect Classification (ODC) provides measurements allowing inferences in the software process by classifying defects according to their types and the conditions that trigger them. A correlation between each type of defect and a phase of the SDP can then be established providing a feedback mechanism pointing to potential problems in a specific phase of the SDP. Chillarege, et. al. state “ODC essentially means that we categorize a defect into classes that collectively point to a part of the process that needs attention ...” [3]. In this way, ODC provides a mean term between statistical defect models [4] and causal analysis models [5].

The classification of defects must be as orthogonal as possible in order to minimize the classification errors. The classification should also be product independent and consistent with all the phases of the SDP. Though ODC has been successfully applied to many projects [3], it should be noticed however, that some controversy still exists regarding the success of ODC in achieving these goals. The meaning of each type of defects and the phase of the SDP associated with them is not presented here but can be found elsewhere [3].

The number of defects of a specific type for a phase of the SDP is expected to follow a certain distribution. Non-conformance with this distribution indicates a potential problem in the process. A sample of expected distribution of types of defects for three phases of the SDP can be seen in Figure 1. As noticed from Figure 1, *Documentation* and *Function* defects are expected to be high for the design phase and decrease as the process moves to the later phases. *Interface* defects are expected to have the opposite behavior.

4 Development of an ODC Demultiplexer

The distribution of the different types of defects at different phases of a software development process can be characterized by a normal distribution $f(y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y-\mu)^2}{2\sigma^2}}$,

where the shape and concentration of errors are determined by the expected mean μ and the corresponding standard deviation σ . The closer μ is to zero the larger the number of defects of type 1 and 2; and the closer μ is to 7 the larger the number of defects of type 6 and 7. The standard deviation σ determines how the different types of defects are spread out for a specific mean. Figures 1(a), 1(b), and 1(c) show the expected defect distribution for a development process. Defects of types 1 and 2 occur more in the Design phase ($\mu = 0.5$). Types 3, 4, and 5 are more concentrated in the Code and Unit Test phase ($\mu = 3.5$). Types 6 and 7 are the majority in the Integration and System Test phase ($\mu = 6.5$).

Defects are inserted into a software product during the many steps of the development process. At a certain level, it can be said that different types of defects are multiplexed (combined) into the software product over its development. The inspection or the test process does not distinguish in which phase a defect was inserted, i.e., they do not serve as a demultiplexer. A demultiplexer is a device that separates signals, in this case defects, that have been combined. ODC can be seen as a demultiplexer by associating orthogonal types of defects with previous phases of the development process. Assume that defects found during the inspection or during the test of a software product are classified based on ODC. This fact can be seen as the separation of two or more channels previously multiplexed as observed in Figure 2.

As stated before, Figure 1 shows the expected defect distribution in a software process. However, non-expected results are more frequent in the software process than one would like. What would be the expected defect distribution in the subsequent phases if an abnormal distribution is observed in one phase? Figure 3 shows an example of the design phase (Figure 3(a)) and one possible propagation of the effects on the Code and Unit Test Phase and the Integration and System Test Phase, respectively in Figures 3(b) and (c).

A solution for the ODC demultiplexer with regard to the state variable model is the integration of the expected distribution into the model output. That is, assume that σ and

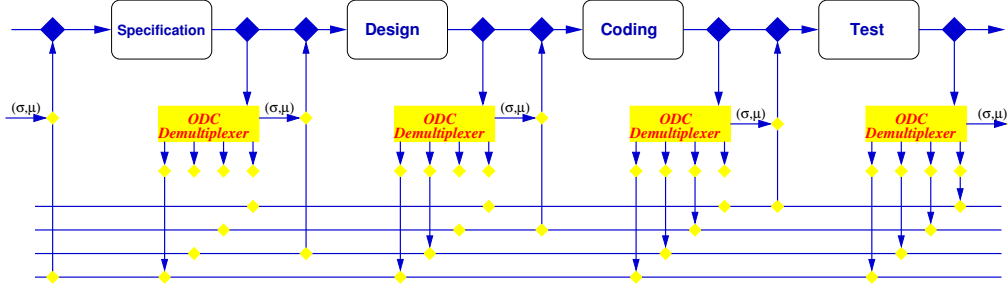


Figure 2. ODC Demultiplexer

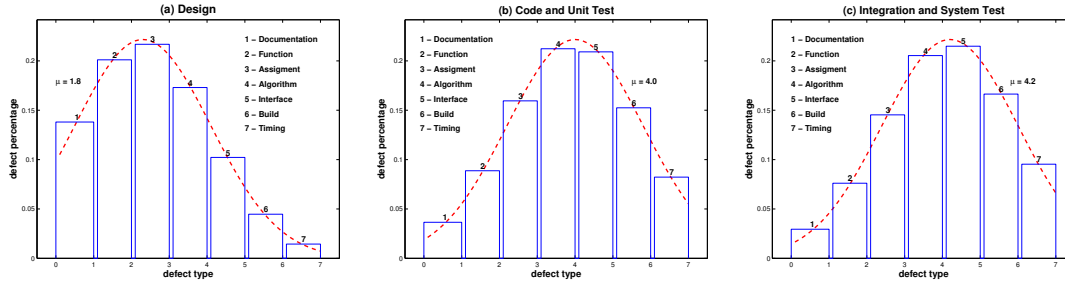


Figure 3. Defect type distribution associated with four phases of the SDP [3].

μ are known for one phase. These values can be used to determine the distribution of defects by embedding a normal distribution function into the output equation of the state model. New values of σ and μ are then computed and propagated to the next phase. The computation of these new values depends on the organization and quality of the development process.

Assume now that observed values for σ and μ are different from the expected values representing an abnormal behavior. Since the computation of σ and μ is done inside the state model, feedback can be applied to regulate the system and make it converge to the desired behavior. Again, feedback application determines what are the necessary changes to achieve the desired results. This process can be repeated every iteration of a development process. Due to its flexibility, these constant changes appear to be more suitable for an Agile Process than to a “regular” software process.

5 Experimental Results

A model with three phases of the development process has been created using Simulink, a MatLab package for simulation and analysis of dynamic systems. The ODC demultiplexer uses a simple approach of splitting the output of the state models according to a pre-specified distribution with a certain variation inserted by a random number generator. Defects associated with the current phase are assumed to be corrected, while the other defects are sent to the corre-

sponding phases. Using the distributions of types of defects presented in Figures 1 and 3 result in the plots in Figures 4 and 5, respectively. As can be noticed, the defects converge much faster for a regular process than for a process presenting anomalies according to ODC. The first stabilizes around 40 days while the second presents a localized stabilization around 55 days. The stopping criteria for the simulation runs is the total number of generated defects. This justifies the convergence to zero for an asymptotically stable process (Figure 4) and the variation over zero for a locally stable process (Figure 5). The convergence also represents the saturation with respect to the accumulated number of detected defects.

The problem with the abnormal process is not necessarily in the defect detection but, most likely, in the actual development of the software artifacts. Under this circumstance, a correction in the detection mechanism will have limited impact on the overall behavior of the process. A more effective improvement can be achieved by delivering artifacts that present the majority of defects associated with the phase it has been developed rather than previous or later phases of the development process. That is, a solution lies in moving the defect distribution of the abnormal process toward the desired values according to ODC signatures. In practice, after collecting data from initial iterations of the process, defect distribution (based on ODC) for each phase can be identified and plugged into the model to predict the behavior of the process. Phases presenting an abnormal behavior can then be corrected to improve the process.

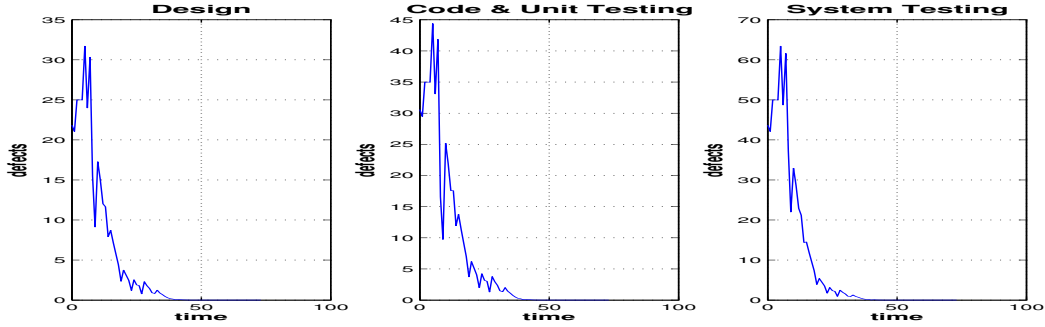


Figure 4. Defect decay behavior for a regular development process.

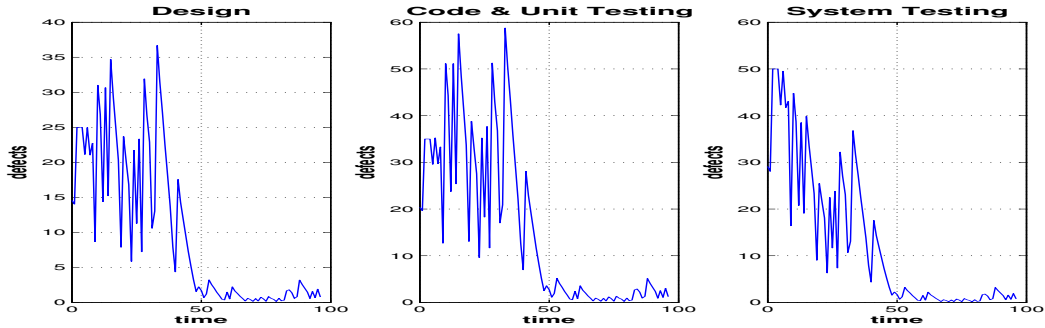


Figure 5. Defect decay behavior for an abnormal development process.

The ODC Demultiplexer from the model has been implemented outside the state model. This prevents the use of a direct adaptive control approach to compute the parameters of the desired distribution. This problem can be solved by incorporating the demultiplexer as part of the output equation of the state model. However, even if the parameters of the distribution can be appropriately computed, we still need to relate the values of the parameters with actual events on the development process. This association along with the insertion of the demultiplexer inside the state model are deferred here to future work.

6 Concluding Remarks

A requirement for the use of the state model for the testing process to other development phases is the presence of inspection/testing gates at these phases. Agile processes present such characteristics and were therefore considered here. However, any ‘spiral like’ or test driven development process with similar features can also benefit from the predictions of the model. Additionally, an ODC filter/demultiplexer is used to properly identify the phases associated with the defects. This helps in determining the feedback path of the defects. Though actual data from projects are required to validate the proposed approach, the results of a model developed in Simulink are a initial indi-

cation of the applicability of the approach. Also, as stated before, the model needs further improvements and the correct association of the parameters of the defect distribution to actual events on the development process needs to be investigated.

References

- [1] A. Cockburn, *Agile Software Development: Software Through People*. Addison Wesley, 2001.
- [2] J. W. Cangussu, R. A. DeCarlo, and A. P. Mathur, “A formal model for the software test process,” *IEEE Transactions on Software Engineering*, vol. 28, pp. 782–796, August 2002.
- [3] R. Chillarege, I. S. Bhandari, J. K. Chaar, M. J. Halliday, D. S. Moebus, B. K. Ray, and M.-Y. Wong, “Orthogonal defect classification - a concept for in-process measurements,” *IEEE Transactions on Software Engineering*, vol. 18, November 1992.
- [4] C. V. Ramamoorthy and F. B. Bastani, “Software reliability status and perspective,” *IEEE Transactions on Software Engineering*, vol. 8, pp. 354–371, 1982.
- [5] W. S. Humphrey, *Managing the Software Process*. Springfield, MA: Addison-Wesley, 1989.