The A-Shaped Model of Software Life Cycle

Csaba Szabó, Ladislav Samuelis

E-mail: Csaba.Szabo@tuke.sk, Ladislav.Samuelis@tuke.sk
Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University in Košice, Slovakia

Abstract: This paper introduces a new, so called A-shaped software life cycle (SWLC) model. We analyze its advantages and disadvantages and highlight the role of the tests in this model. Later, we show the granularity of the incremental development and its impact on software evolution. Finally, we discuss the parallel feature of this model.

Keywords: A-shaped model, incremental development, software life cycle, test evolution, V-shaped model

1 Motivation

The waterfall [5, 10] or the V-shaped [11] SWLC models are a sequential path of action (process) execution. Each phase must be completed before the next phase launches. The V-shaped model emphasizes testing and planning testing with a breath of parallelism between planning and design. But none of them enables evolution – incremental development in the test planning or in the application design phase. The concept of evolution is also the topic of the machine learning, e. g. [6].

Our goal is to design a new model for supporting parallelism and evolutionary design of the application and of the test plans too. Test planning cannot be separated of the application design because of its metrics' positive influence by identification of the costs of testing.of each component as shown in the papers [8, 9].

2 Introduction to the Model

This model copies the classical sequence of actions from the waterfall (or V-shaped) model. It stems from the requirements and branches into two processes which end with implementation of the application resp. implementation of the test plans. We introduce here and emphasize the mutual influence between the development and test planning phases based on observation we state that some activities may be executed in parallel.

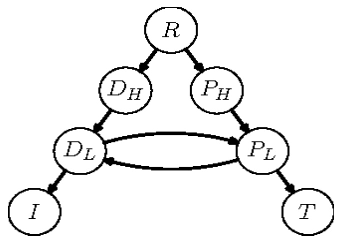

Figure 1
The A-shaped Model

Requirements engineering (R). In this phase the requirements [1, 10] are gathered and preprocessed in a way of their separation into two sets. One of these sets is the basis for the application development, the another one is the basis for designing the tests.

2.1 The Process of Design

Design – high level (D_H). This phase includes the implementation of the functionalities at the highest level of abstraction.

Design – low level (D_L). In this phase we refine the ideas from the higher levels iteratively and incrementally. In separated cases, the test planning process may produce tests due to those some selected elements of the design may change – test driven development of the functionality [2, 4].

2.2 The Process of Planning

Planning – high level (P_H). This phase is about functional test planning. Test plans are prepared for testing functionality and are decomposed (hierarchically) at this level.

Planning – low level (P_L). The detailed planning of tests and dependency analysis follows the decomposition. After the development of the structure of functionality and design oriented tests, a new functionality can be introduced via inclusion of a new functionality oriented test or via a new design element at the selected level of abstraction. In other words, inclusion (or deletion) of functionalities may be executed in both test driven [2, 4] and classical (in the design process [3]) way.

2.3 Implementation

Implementation (I). This is the final phase of refinement (design) where all the coding takes place.

Test implementation (T). The test plans are implemented in the form of a program, or other testing code.

2.4 Description of the Phases and their Results

By the analogy with the waterfall model, we distinguish between R- (from Requirement), H- (from High level), L-phases (from Low level) and implementations in this model.

2.4.1 R-phase

The first phase belongs to the requirement engineering [1, 10]. The outputs are two subsets from the perspective of their usefulness in the design and testing. During this process the requirements are decomposed and categorized. Categories serve for better requirement tracing and separation.

Two phases follow after the R-phase in parallel: higher level design and planning of testing.

2.4.2 H-phases

At the higher level, we can see two phases executing in parallel. Both of them are based on functional and structural decomposition as refinement activities.

Higher level design interprets architectural ideas of the system being developed, higher level planning outputs functional testing concepts at a very high level of abstraction – the basic structure of the upcoming tests on both architectural and behavioral base.

Both phases result into their lower level correspondents.

2.4.3 L-phases

The core of the SWLC model is built up from these phases, D_L and P_L (the L-phases). It is the point where the parallel threads are synchronized. These two phases may be executed in parallel but there is a significant influence between them that makes the core of the method incremental (we describe it later in sections 3 and 4).

A detailed model of the system is designed in the phase that includes the full architectural and behavioral specification of the system in the modeling language selected by the designer. The whole model is built from the results of the higher level design using similar refinement steps. The only difference is, that all data are specified here with the design of the operations with them.

The phase results into the model of the tests, the behavioral and architectural specification of tests, test contexts and data. The test cases are refined to the crisp values and dependency definitions (e.g. which design element is tested by which test case). These results come from the stepwise refinement of higher level test specifications.

2.4.4 Implementations

Implementations (the final system and the implemented tests) are generated during the whole development many times as prototypes. These implementations are the outputs of the actual models at the L-phases. The line between the design and the implementation is clear: the point of applying a concrete, programming language specific aspect. The mentioned border is that between the portable and special architecture.

Further the implementation of the system is stressed against the corresponding tests in the testing procedure.

2.5 Features

The features (as always from a certain aspect) can be divided into two groups [3], e.g. advantageous and disadvantageous ones. In the next sections, we analyze them.

2.5.1 Advantageous Features

- Design is split into levels (D_H , D_L), the lower levels are developed by refinement steps.
- D_H is parallel to P_H , so the requirements can be processed in parallel (for tests and for the application).
- Tests are developed by refinements from the highest to the lowest level that offers better portability of the tests (test models).
- D_L and P_L influence each other by offering new requirements for the other (parallel) thread of execution, that makes the process incremental.
- Requirements are divided into two groups during the engineering phase that is the prior factor of categorization. These output sets may be further divided into subsets of different categories, e. g. priorities.

- The model fully supports requirement tracing and through the documented relations between the design and test elements an extended feedback to the design too.
- Due to the fact, that the system and test implementation phase are executed in parallel, the delivery of the application and the tests for it could be synchronized – this can shorten the SWLC time.

2.5.2 Disadvantageous Features

- The parallel execution of the D_L and P_L is determined by the mutual influence between these phases.
- The dependency detection procedure could become very complex at the beginning of the development while detecting the relations between the base sets of design and test elements.
- The SWLC model does not provide a notation or a set of patterns to increase reusability.
- There are no explicit mentions e. g. about the missing maintenance and other SWLC phases.

3 Remarks on the Evolution of the Tests

The model of the tests includes records about the used (tested) design elements which are the traces for the change propagation or just for the dependency monitoring.

These records allow to define a change propagation across both models.

Changes caused by design activities propagate changes in the test model in the form of a changing requirement what and/or how to test. In this way, change propagation is done by changing the initial requirements for the affected tests. The adaptation process (to the change) is de facto started as introduction of a new requirement or a modification into the requirement set. Looking at it from a wider perspective, there is an evolution inside the SWLC model.

This idea works vice versa for the design thread.

4 Remarks on the Parallelism

Considering the two outputs from the R-phase (no influence between L-phases) we can split the activities into two groups.

The design thread is than the same as in the waterfall model. The test development represents as a separated development of the testing application.

The parallel threads allow an independent design of tests and the application, but without the joining of them we loose the ability to design complete tests, e. g. tests specialized (passed) to the designed components of the system. On the other hand, joining of the test planning thread with the application design one provides the ability of test driven development [2] of some parts of the system.

Conclusions

We showed the A-shaped SWLC model and its pros and cons. This model covers the test planning phases of the SW development, it shows the location of these phases and the dependencies between them and the design ones.

The development of both the application and the tests is model driven that increases the portability of the solution being developed.

The model covers the evolution of the tests via considering the design as the extension of the requirements set for test design and planning. It may include the possibility to generate test cases to the design [7], but with the extension to map the relations between these test plans and tested design elements (and the tested functionality too).

The next step in the development of this model is the extension of the abstract system dependency graph [12] by the test plans and the requirement hierarchy and putting it to a higher level of abstraction (considering not only classes as elements).

Acknowledgements

The research was supported by the following grants:

- Mathematical Theory of Programming and its Application in the Methods of Stochastic Programming. Scientific grant agency project (VEGA) No. 1/2181/05
- Technologies for Agent-based and Component-based Distributed Systems Lifecycle Support. Scientific grant agency project (VEGA) No. 1/2176/05

References

[1] Jim Arlow, Ila Neustadt. *UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design*. Addison-Wesley, 2nd edition, June 2005, ISBN: 0-321-32127-8

[2] Kent Beck. *Test Driven Development: By Example*. The Addison-Wesley Signature Series. Addison-Wesley, 2003, ISBN: 0-321-14653-0

[3] Douglas Bell, Ian Morrey, John Pugh. *The Essence of Program Design*. Pretince Hall Europe, 1st edition, 1997, Hungarian translation: *Programtervezés*, Kiskapu Kft., 2003, ISBN: 963-9301-57-4

[4] Chromatic. *Extreme Programming Pocket Guide*. O'Reilly Media, Inc., 1st edition, July 2003, ISBN: 0-596-00485-0

[5] Zdenk Havlice. *Modelovanie a prototypovanie informačných systémov*. elfa, s.r.o., January 1999, ISBN: 80-88786-95-9

[6] Kristína Machová. *Strojové učenie. Princípy a algoritmy*. ELFA, s. r. o., Košice, Slovakia, 2002, ISBN 80-89066-51-8

[7] Jeff Offutt, Shaoying Liu, Aynur Abdurazik, Paul Ammann: Generating Test Data From State-based Specifications. *The Journal of Software Testing, Verification and Reliability*, 13(1):25–53, March 2003

[8] Zoltán Porkoláb, Ádám Sillye: *Comparison of Object-Oriented and Paradigm Independent Software Complexity Metrics.* ICAI'04 6th International Conference on Applied Informatics, Ed. Lajos Csőke et al. Eger, 2004, pp. 435-444

[9] Zoltán Porkoláb, Ádám Sillye: *Towards a Multiparadigm Complexity Measure.* 9th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering QAOOSE Workshop, ECOOP 2005, Glasgow, pp. 134-142

[10] Ian Sommerville: *Software Engineering*. Addison-Wesley Publishers Ltd., Pearson Education Ltd., Boston, MA, USA, 7th edition, 2004

[11] Csaba Szabó: The V-shaped Model from the Testing's Point of View. In *Proceeding from the 6th PhD Student Conference and Scientific Competition of Students of Faculty of Electrical Engineering and Informatics, Technical University of Košice*, pp. 127-128, Košice, Slovakia, 2006, elfa, s.r.o.

[12] Z. Yu, V. Rajlich: Hidden Dependencies in Program Comprehension and Change Propagation. In *Proc. International Workshop on Program Comprehension*, pp. 293-299, IEEE Computer Society Press, 2001