Proceedings of the 7th International Conference on Applied Informatics
Eger, Hungary, January 28–31, 2007. Vol. 2. pp. 309–316.

On the role of incrementality in test design*

Csaba Szabó[a], Ladislav Samuelis[a], Roman Bazelides[b]

[a]Department of Computers and Informatics
FEEaI Technical University in Košice
e-mail: {Csaba.Szabo,Ladislav.Samuelis}@tuke.sk

[b]Faculty of Electrical Engineering and Informatics
Technical University in Košice
e-mail: bazdic@gmail.com

Abstract

This paper deals with the abilities of consolidating the system design and test planning phases of the software life cycle (SWLC). After the description of the motivations and theoretical background, the introduction of the A-shaped model of SWLC follows with emphasis on the change propagation across the design and test plan. Further, related features of the model are discussed and conclusions end the paper.

Keywords: A-shaped model, change propagation, incrementality, software design, software life cycle, system dependence graph, test plan

MSC: 68N19

1. Motivation

Programmers has been searching for repeatable, predictable methodologies, or processes that improve productivity and quality of the software design, for several decades. Writing complex software is seemingly unruly task and requires careful application of formal as well as managerial tools.

The work outlined in this paper is motivated by the necessity to speed up the whole process of the inclusion of a new functionality into the system, we call it speeding up the incremental change.

Programmers often have to modify definite parts of the system under development, or just extend it by a new component. In the case when a certain component is proposed for modification and it influences other parts of the system, then it is

*This work was supported by project VEGA No. 1/2176/05.

necessary to modify the influenced components too. This process generates a set of systematic regression changes, which is elaborated in [10]. After the implementation and/or inclusion of all required changes, the regression testing of the system follows in order to ensure whether the functionalities of the system remained.

Because of the aims to shorten (minimize) the duration (execution time) of the software projects, it is useful to design an approach to speed up the phases of evolution and test preparation for regression testing. This paper offers an idea how to increase the efficiency and decrease time of execution of these phases by consolidating the design and test planning phases using the incremental change approach.

2. Background

One of the main issues of the software development is to wisely combine general and specialized methods in order to solve the task effectively and efficiently. In other words, combinations of selected methodologies and tools at software systems development are prerequisites for successful delivery of the project.

2.1. Methods and models

Lots of SWLC models and related methods are discussed in [3, 8]. Some of them are more universal, others prefer early release over flexibility. Table 1 presents examples of SWLC methods and/or models without claiming for completeness.

• Waterfall • Staged • Spiral • Incremental • Unified Process [1]	• Extreme Programming (XP) • Feature Driven Development (FDD) • Test Driven Development (TDD) [2] • Full Life Cycle Object-Oriented Testing (FLOOT) • Agile Model Driven Development (AMDD)
"Classical" methods/models	Agile methods

Table 1: SWLC models and methods

Remark 2.1. There are always requirements in the middle of interest, but these are processed and taken into consideration in different ways.

2.2. Top-down vs. bottom-up development

There are two main approaches in software development that can be applied in the methods mentioned above: top-down and bottom-up development.

Top-down (TD). From *requirements* to *bytecode* is told here – the requirements are one-by-one fulfilled by specialized components (model driven).

Bottom-up (BU). Reusing of *bytecode* to fulfill the *requirements* is the core of this approach – more universal components are combined to meet the requirements (COTS Components Of The Shelf). In this way the unwanted functionalities can appear in the SW.

Remark 2.2. The best practical solution seems to be to find the point where the BU and TD approaches intersect and fuse (are woven).

3. Tests and test design

Testing phase belongs to the base phases within each SWLC approach let it TD or BU. Either approaches are less or more so-called agile.

Testing is important *before* any release. This is the claim of all release-oriented methods. Others claim that testing may be independent from the release phase and it may be provided as often as possible. In all cases, the test efforts have a high relevance to the release and documentation.

For testing, first, we need to have something to test. A (not necessarily release) version of the application is processed against the existing tests that can cause new tests to appear or cause that a part of the existing solution have to be rewritten.

In order to run tests effectively, we need to have a test plan. The more revised plan we have, the better. To get one, we need to design it – that is the test design.

3.1. Test design

Test design is a phase and a model too. From the point of view of the development process, it is a group of activities around test preparation: planning, generating or developing tests. On the other hand, it is a model that may have its sophisticated structure according to selected standards and it may be documented as well.

3.1.1. Test preparation

Test planning gives answers for the questions: *what? when? why (not)? how? who?* It is the classical meaning of planning. Test preparation includes all tasks according to the creation of the test elements, that means, the concretization of the *what-how-when* triangle is being processed here.

The automated way of the test creation is the test generation [7], that can be made from the application model or in special cases from the source code.

3.1.2. Test structure and documentation

Use of standards in the testing phase is important from the perspective of maintainability of the tests. Common standards or categories are as follows:

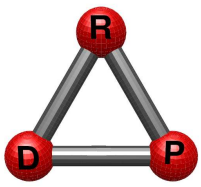
- nothing,
- company standard,
- programming standard (JUnit [4]),
- TTCN/UML TP [6],
- IEEE 829–1998 [5].

These standards describe the expected structure as well as the way of documenting the testing process. In most cases, their combination is applicable to increase the efficiency.

All above-mentioned standards recommend a structure that can be implemented in a specific form of a system dependence graph (SDG) analyzed by Yu and Rajlich in [10].

3.1.3. The place of the test plan

Any representation of the tests (a test plan or test model) includes a description or a list of the parts of the system (as parts of the system design model) that have to be tested, at least at a low level of abstraction near to the implementation of the tests. Considering that and the fact that the parts of the system being developed are related to the required functionalities (the requirement model) as well as the tests are, we can find a triangular or cyclic dependence between these three models being processed during the SWLC. E.g. at the top, we can place the requirements, below that, we find the place for the design and test models and the relations between them. Figure 1 shows these places.



R: Requirements model.

D: Design model.

P: Test plan/model.

Figure 1: The triangular relationship model

4. Incrementality in test design

Analyzing the character of the relations between the test, design and requirement models, we draw the conclusion that these relations are bidirectional, e.g. requirements determine what and how to test, but tests help to better understand the requirements as well as it is by the combination of requirements with the design model. The third combination is more important. Tests and the design are in the early phases of SWLC independent, but in the lower level designs and more precise test specifications these two models become strong dependent on each other, and depending on the development method one of these models can be the master model determining the second (slave) ones structure and/or behavior. E.g. in TDD the test model (represented as primary test implementation) is the master and the design one (the application source code in our case) is the slave.

In all cases, extension of the design model or insertion of a new requirement could break the relations between the changed model and the tests associated with it. Any change inside these related models causes a possible change inside the test model. This phenomenon is the incremental change in the test model triggered from the application design model, e. g. there is incrementality in the test design.

In [9] we introduced the A-shaped model of SWLC the structure of that is shown on Figure 2. This SWLC model deals with the mentioned problem as well as with the incremental changes inside the design model.

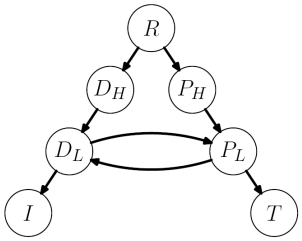


Figure 2: The A-shaped model

Levels of abstraction and the granularity of the elements may differ in general inside each so-called submodel (if considering the triplet as the whole model) of the triangle on Figure 1. What remains is that requirements are the basis for everything. High level design (D_H) and test planning (P_H) represent the high level of the top-down approach. Low level design (D_L) and planning (P_L) are at the lower level, but the more important factis that the incrementality plays there the the main role. Implementation (I) and test (T) component binaries are the final level of the TD development, or they may help to use the bottom-up approach by including a component off-the-shelf.

At the H-level (D_H, P_H) there is no need to know the structure of the second sub-model, at L-level(D_L, P_L) we need to know about it, therefore the execution

of the phases is as follows:

- at H-level – D_H and P_H may execute in parallel and work separately with the models D and P ('P' is for test **P**lan)
- at L-level – D_L may raise changes in the model handled by P_L and vice versa that is the incrementality across the group of design and test models in test and application design processes, see Figure 3.

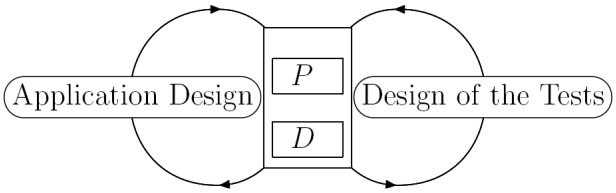


Figure 3: The principle of incremental change in the design and test models

Incrementality in the P_L phase needs no regular incrementality on the D model being provided, only a simple non-incremental change such as extension, refinement of the design model being developed could cause a big change (respectively a group of smaller, mostly incremental changes) in the model representing the test structure and procedures. The facts substantiating this behavior are as follows:

1. The model of the tests includes records about the tested design elements which are the traces for the change propagation or for the dependency monitoring.
2. The records allow to define a change propagation across both models handled by P_L and D_L.
3. Changes caused by design activities propagate changes in the test model in the form of a changing requirement what and/or how to test.
4. From wider perspective, there is an evolution inside the model of the tests.

We can build the SDGs for each submodel that have generally different granularities of their building components. These sub-graphs will be further interconnected by edges representing the existing relations between them. The final digraph will be the model of our system being developed at the selected levels of abstraction and will serve for change propagation simulations for this system. An example graph is shown on Figure 4, the black colored edges represent the classical relations inside each model needed for change propagation simulation, the blue colored ones are the relations between the corresponding elements of the submodels and serve for the realization of the evolution in the test model.

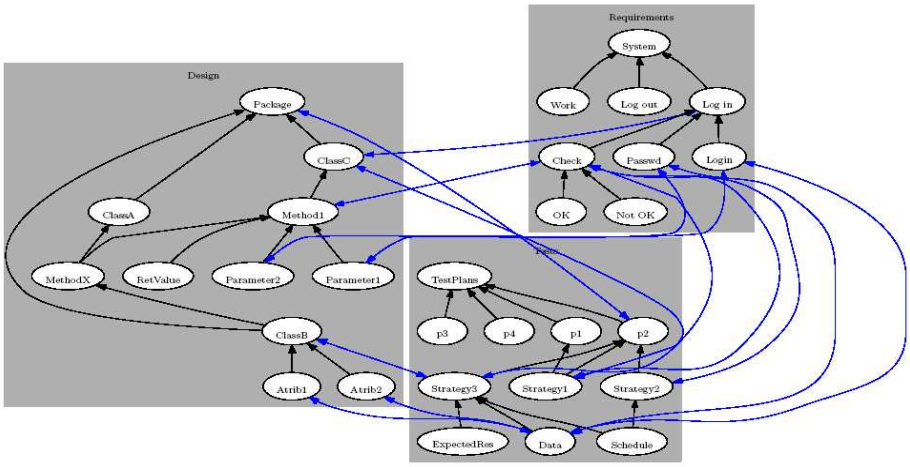


Figure 4: Graph representation of the relations between the models

5. Conclusions

After introducing the most common tasks and features of methods of the software development, we discussed the ways of thoughts on testing and test design.

We showed the A-shaped SWLC model that covers the test planning phases of the SW development, we showed the location of these phases and the dependencies between them and the design ones.

The development of both the application and the tests is top-down that increases the portability of the solution being developed because of the created models.

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).

In this paper we discussed mostly the relations between the test and the design models, but there are two more interesting relations which are ones between the requirement and design models (we guess the most discussed topic from these three ones), and ones between requirement and test models.

The next step in the development of all algorithms for this model as the extension of the system dependence graph [10] for the test plans and the requirement hierarchy and putting it to a higher level of abstraction (considering different granularities in the submodels).

References

[1] Arlow, J., Neustadt, I., UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design, *Addison-Wesley*, 2nd edition, (June 2005).

[2] Beck, K., Test Driven Development: By Example, *The Addison-Wesley Signature Series, Addison-Wesley*, (2003).

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

[4] Gamma, E., Beck, K., JUnit, Testing Resources for Extreme Programming, (24 November 2005), `http://junit.org/index.htm`

[5] IEEE Standard for Software Test Documentation, IEEE 829–1998, (1998).

[6] Object Management Group. UML Testing Profile, Version 1.0, (July 2005).

[7] Offutt, J., Liu, S., Abdurazik, A., Ammann, P., Generating Test Data From State-based Specifications, *The Journal of Software Testing, Verification and Reliability*, 13(1) (March 2003), 25–53.

[8] Samuelis, L., Szabó, Cs., Notes on the role of the incrementality in software engineering, *Studia Universitatis Babes-Bolyai Informatica*, 51(2) (2006), 11–18.

[9] Szabó, Cs., Samuelis, L., The A-shaped model of software life cycle, In *Proc. of the 5th Slovakian-Hungarian Joint Symposium on Applied Machine Intelligence and Informatics, SAMI 2007*, Poprad, Slovakia, (January 2007), 129–135.

[10] Yu, Z., Rajlich, V., Hidden dependencies in program comprehension and change propagation, In *Proc. International Workshop on Program Comprehension*, IEEE Computer Society Press, (2001), 293–299.

Csaba Szabó, Ladislav Samuelis, Roman Bazelides
Department of Computers and Informatics
Faculty of Electrical Engineering and Informatics
Technical University in Košice
Letná 9
04200 Košice
Slovakia