INFORMATION
SYSTEMS IN
MANAGEMENT

Information Systems in Management (2016) Vol. 5 (1) 88−98

MODELS OF IT PROJECT MANAGEMENT IMPLEMENTATION AND MAINTENANCE

ANNA PLICHTA [a], SZYMON SZOMIŃSKI [b]

[a] *Cracow University of Technology, Department of Computer Science*
[b] *AGH University of Science and Technology*

The process of software development has become so dynamic that it nowadays requires more and more supporting tools. An organization which intends to have such tools implemented must take into account its possible future needs. Complex solutions usually offer greater variety of available options and functionalities but are costly to implement. This paper presents some models o IT project management together with the supporting software. Owing to applying the resource management policy at the stage of production and implementation one may assess the risk and identify potential threats. The discussed issues may shed some light on potential difficulties occurring at the particular stages of software production management whereas the conclusions may help people in charge choose the best model, including its implementation strategy. The first model are dedicated tools created from scratch and often preceded by the analysis of the existing solutions and the needs of the company and the customers. The second is to seek from the very beginning multifunction and complex tools for the big or well-developed IT systems providers. This article is to present the models regarding the IT project management (together with the related supporting tools) which are applied in the IT companies involved in providing the foreign customers with the bases of knowledge. On the basis of the experience gained while designing and developing dedicated software (by means of the above-mentioned tools) and its implementation one drew some conclusions concerning e.g., risk assessment; potential threats at every stage of the project lifecycle; improvement of the quality and production time of the software; reduction of the number of errors; improvement of the internal communication within the project team (it is the first step in the development of the design patterns). The patterns should help the managers choose the proper management model and related tools for the implementation and particular project tasks.

Keywords: project management, risk assessment, software life cycle, software implementation

1. Introduction

Many IT tools supporting the project management is currently available on the market. Some of the are one-task applications (e.g., dedicated to generating project timetables or reporting), some are complex, multi-function ones for project portfolio management in the entire institution.

Growing demand for the creation of new and more complex software is no longer unusual. Everyone who took part in its manufacture is aware of the complexity of its production .These are: the specification of the functional and system requirements that meet real needs and expectations of the customer; designing and implementation of the system in accordance with the specification (including the tests proving that such an accordance was met); the ability to be updated and modified. Hence, the proper organization of the work, especially good design patterns and project management tools are very important to effectively produce the software. The application of the verified methods and good practices may solve many typical IT issues and lead to the creation of high-quality, scalable and flexible code.

Until recently, one person could deal with the process of gathering requirements, analysis, design, programming, testing and implementation of the created solutions. Exploratory programming is not a good practice for complex software production on a massive scale. The complexity of the currently created systems is simply too big for a single person to maintain throughout the whole life cycle.

The difficulty resulting from the size of the system makes it necessary to standardize software development process. Over the years there have been several models of action and states in which an IT product currently is. A set of models of software life cycle is vast. We can highlight some key models, while the rest of them are mostly a combination of two or more of related methods.

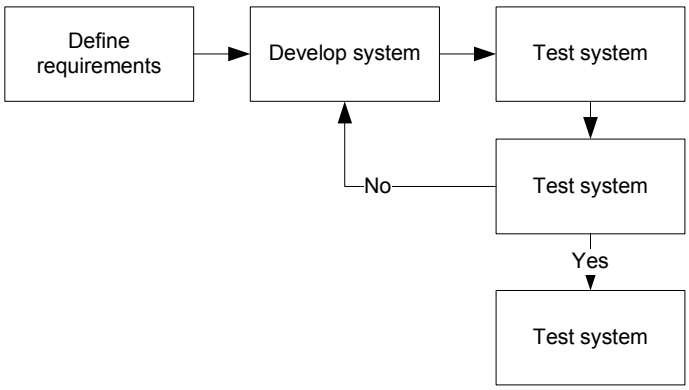

Figure 1. Exploratory programming

The software life cycle model consists of a series of interdependent phases. In one of the first steps one defines the needs for the construction of an information system by means of clarifying the idea of its construction, operation, parameterization resulting from the nature of its activities, and ending with the cessation of its operation. In this paper we will present those parts of the software life cycle that are directly related to the implementation management process and IT project and its successive maintenance.

The most common models used during software development are the cascade model and iterative model. In each of these models we can distinguish some phases describing a set of actions whose aim is to create a working system that meets the client's expectations.

2. Life cycle models system

The cascade model also known as cascading waterfall model is the oldest and most famous model of the IT product life cycle management. This model is commonly used because it seems to be the most natural one. Here, the problem is divided into several consecutive steps. Anyone who applies the waterfall model to build an IT system should go step by step through all its stages in a strict order. The phases that can be distinguished in the cascade model are among others: the gathering of requirements, analysis, design, implementation, testing, implementation of the entire system.

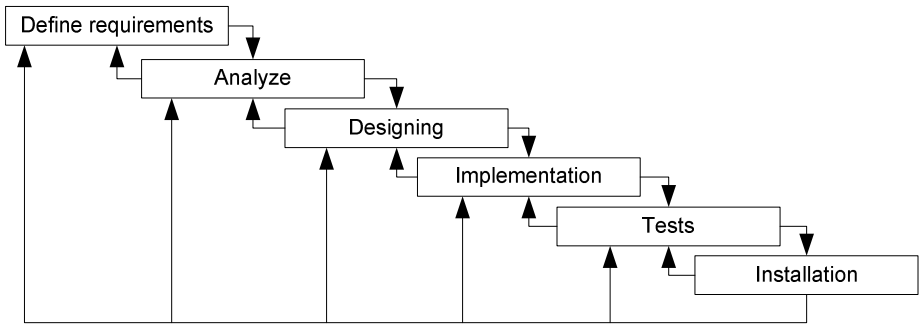

Figure 2. Cascade model

In this model, the output of one stage is a point of no return: one cannot make any step backwards. Thus, the sequence of stages is rigid. After creating a model of the problem domain an analyst designer passes the solution to a designer who in turn accedes to create a software project. Then, the programmer is responsible for

the implementation. The next step is a verification of the implementation intended to eliminate the mistakes so that one can deliver the product to the customer.

- The order of transfer of results in cascade model is very important. Indeed, sometimes a return to the already completed phases of the model is necessary, but such situations should be avoided. The verification at each stage is therefore inevitable, especially when the team encounters errors at the stage of implementation. The waterfall model has the following features:
- Obtaining a product meeting customer expectations is very strongly dependent on the stability of requirements which are, in fact, very difficult to clarify at the beginning of designing.

The phase of verification of the product compliance with the requirements is carried out only in the final stages.

- If one tries to personalize the product in response to the change of requirements, the cost of the system creation must inevitably rise.
- The order of execution of the work must be strictly followed, but it is not necessarily a disadvantage.
- The high cost of mistakes made in the initial stages is very characteristic feature of the cascade model. The error at the stage of collection or analysis of requirements can be detected only at the stage of acceptance tests or, which is even worse, during the operation. The costs of their removal may significantly exceed the costs of mistakes made at the implementation stage.

The waterfall model, despite its peculiar features, in a slightly modified form has become a standard recommended in the development of software for military use. According that modification, after each stage of development a series of documents is created and only their approval is sufficient to move to the next stage of the software development.

The Iterative model organizes the requirements and divides them into smaller subsets. Each distinct subset contains a set of defined functionalities. They must be first executed, which enables for the transition to the next set. Completed iteration provides a partial functionality that has passed through all the stages of software development.

This approach allows you to divide the project into smaller parts and to execute them according to the established priorities thereby eliminating the risk of the failure of the whole solution. Iterative model enables for the very fast verifications of the feasibility of the developed system and for gathering the feedback from the customer. In order to obtain an overall picture of the scope of the functionality of the designed system one conducts a process called requirements reconnaissance. The result of this process is the assignment of the specific tasks to the particular iterations and making a schedule of their implementation. These pre-defined

actions rely mostly on developing an overall architecture of the system, and sometimes even its prototype.

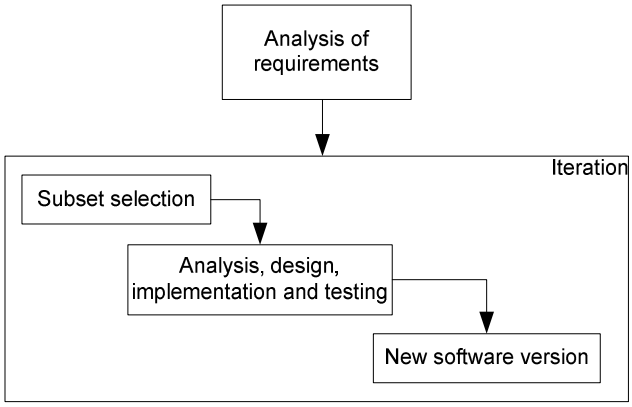

Figure 3. Iterative model

One variant of the iterative model allows for the implementation if a part of the system has been tested and is compatible with the requirements of the customer. This implementation brings both financial benefits associated with the partial implementation of the system, and allows for getting fully objective opinions about its performance in the real-world conditions. The implementation of the system takes place in several steps where each step contains a certain subset of completed iterations.

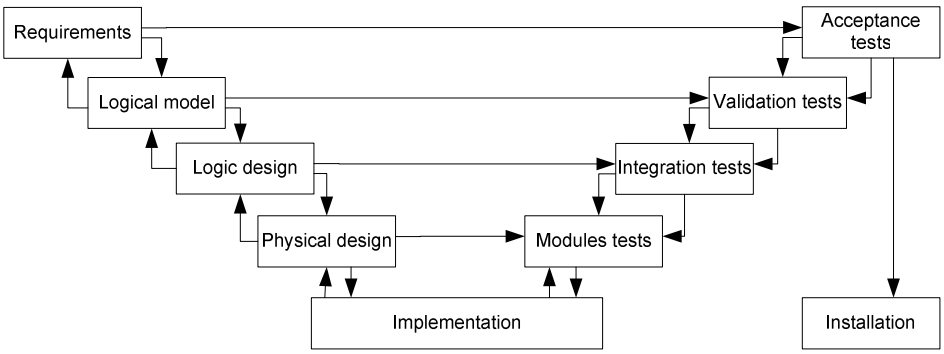

Figure 4. V model

The development of the cascade model is the model V. It gained popularity thanks to the extensive testing phase of the produced system. Tests at every stage of the software life cycle are for the sake of verification and to validate the correct-

ness of operations included in it. Each step has been provided with the testing stage so one gets a product that meets customer requirements.

An attempt to formalize the iterative approach to software development is a spiral model. This approach analyzes the risks that occur in each iteration. Continuous monitoring and measuring changes which are subjected to critical evaluation by users allows one to conduct the risk analysis. One of the first steps within the spiral model is the analysis of the prerequisites. If the requirements seem to be achievable within the prescribed time, budget and the available resources (i.e. the principle of the triangle) one can start the project planning and the first iteration.

Each iteration takes the form of small waterfalls followed by a review of the system. If the project requires further work then one needs to plan the next iteration and perform risk analysis. Spiral model is a variant version of the waterfall model based on the current risk analysis.

In the spiral model, we can distinguish four basic steps included in the system:

- Planning - based on the requirements and objectives set by the customer, it shall be made to identify the alternatives and limitations and for planning iterations each time one starts the next spiral cycle
- Risk analysis - is simply the assessment of alternatives, and attempts to identify and analyze the risks associated with each possible alternative construction of a new solution.
- construction - is in the form of a small waterfall, and its aim is to produce the next version of the system.
- evaluation by the customer - verification of the created solution and its assessment with the possibility of modifications to the requirements on the system (potential modifications to the requirements should be avoided).

The benefits of the spiral model include the mitigation of the risk of failure of the project and product verification by the user at each stage in order to make a product fully satisfying the customer. An attempt to deal with the lack of identification and lack of requirement stability was included in the model of prototyping.

In this model a system is created by building up successive approximations so that each of the subsequent prototypes could reflect the requirements as close as possible. Evaluation of the prototype and subsequent versions of these prototypes in a very natural way leads to the identification of requirements. Prototyping very often coincides with the requirements analysis phase and hence often depicted model is referred to as prototyping requirements. A characteristic feature of rapid prototyping is brisk pace of its creation putting its emphasis on its quality and adaptation to the target environment. A broader view of prototyping in order to contain the design phase in order to verify the effectiveness of the solutions adopted is also possible.

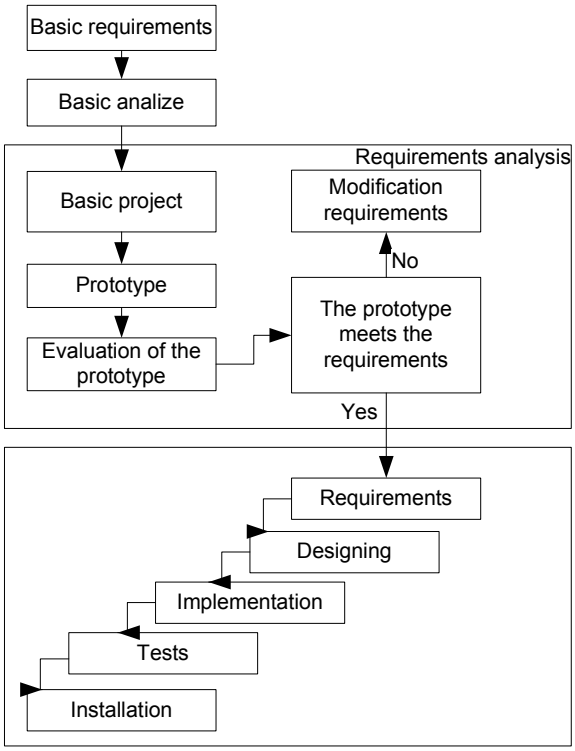

Figure 5. Prototype model

The specific features of the models influence the maintaining and management of the entire IT project. The one who chooses between the particular policies towards the project management should take account of many correlated factors. The cascade model is less risky and more advisable in the case of small projects, because the iterative model is time consuming as it may require the IT specialist to verify the assumptions (regarding the bases of knowledge) many times. On the other hand, the iterative model facilitates implementing all changes, because the client is more involved in it in the process of software production. That model allows for the introduction of new technologies to raise the qualifications of the team members and thus the software quality. The V model and the spiral model, in turn, are based on the standardization of the methods of introduction of the new technologies and related risk in order to make the company more competitive. Each of the discussed models has its pros and cons, so one should choose the most suitable for the particular project. The correct decision may reduce the costs and time of the production. Otherwise, the customer may resign.

3. Tools Supporting Software Development Process

In the process of gathering requirements tools such as Computer Aided Software Engineering (CASE) are helpful. Enterprises Architect is one of them.

Enterprise Architect is one of the most popular tools to support IT projects in the following areas: systems in UML modeling, requirements management, change management and IT project management support. User-friendly interface allows one to quickly start working with the tool, but novice users of the tool may require a lot of time to use its full capabilities

Possibilities of the tools are tailored to different levels of sophistication of users and to the needs and expectations of the client. Among them we can distinguish the following elements:

- developing the project by means of the creation of UML models; creation of all types of diagrams with full specifications including custom diagrams
- creating project templates and models
- configuration of the system to teamwork - a version control system
- creating professional documentation - document management and creating virtual documents
- management model - different types of repositories according to specified criteria of selection (including the configuration procedures and the operations on repositories)
- advanced modeling - the use of patterns and creating one's own design patterns, the use of profiles and creation of their own profiles, using MDA transformations to create specific models
- Project management - change management, testing and effort estimation based on use cases.

Enterprise Architect Professional Edition is a tool for developing complex IT systems. It has been provided with the ability to define a shared database repository for projects as well as to plug project to the selected tool enabling for the version control. Professional version has built-in option of developing code and database and technology support MDG (Model Driven Generation) and the debugging process. It also has full functionality to generate documentation in HTML and RTF. This edition of the tool supports the process of project management and management (tracking) design requirements.

An important issue in the life cycle of the software is the control regarding its version, and the identification of the reasons for the emergence of errors at each stage. Version Control System or Visual Studio Team Foundation Server is a powerful software serving as a groupware server for teams of programmers and for project managers.

TFS server stores the application source code and other project files, offering sophisticated version control, automatic compilation, reporting and project man-

agement. The software integrates multiple server technologies from the Microsoft and configure them so that they are useful in the software development process. The teams may use, for example, SharePoint and some functionalities of the Project software. The biggest advantage of TFS is full and built-in integration with the Visual Studio 2013. Hence, the cooperation between the two programs is running smoothly.

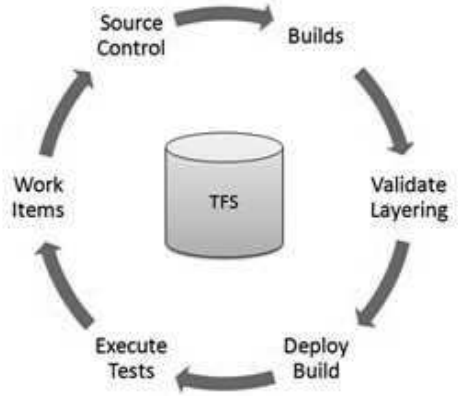

Figure 6. Team Foundation Server workflow

TFS allows you to manage your repositories, compilation processes, testing infrastructure and implementation. It provides you also with the possibility of easy collaboration and status reporting. Program Team Foundation Server (TFS) 2013, created to maximize the performance of development teams, is the center of product lifecycle management for Visual Studio. It allows all persons participating in the project to actively participate in the process of creating software using one of the solutions.

The basic functions of the environment can include, among others:

- Agile planning and cooperation - the use of agile software development at one's own pace, using templates for Scrum, Agile and CMMI®. Downloading process templates from other companies or creation of the own ones. Cooperation with all project participants throughout the development process, using working elements and Kanban boards so that everyone involved in the project could be integrated into the workflow
- Compilation - capturing and analyzing defects and other quality problems at an early stage of application development. Code verification, testing for additional control.
- Test cases management - permanent quality assurance - getting test cases management functions through the web access. Creation and execution of the test cases remotely, participation of all members of the team in the reviews of test cases. Profiling the test units to form a better code via comprehensive tracing of the flow of the code, including the sole test units.

- Reporting - tracking work items, report generation based on the current operating status. Quick access to required information.

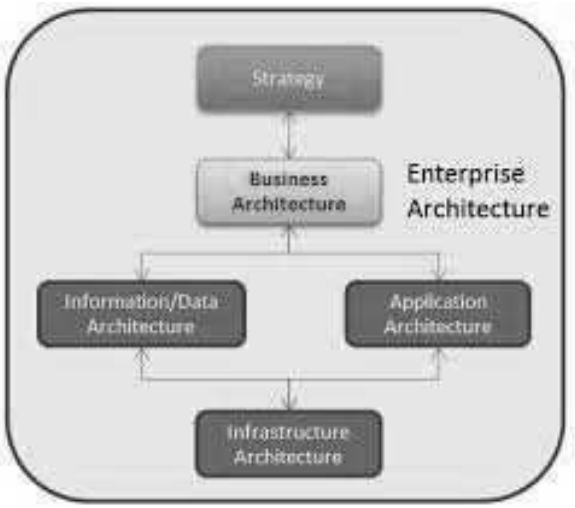

Figure 7. Enterprise Architect

4. Summary

In the article we presented the models of IT project management including the stages of design, testing, implementation and development of good practices during the projects. The paper is general in its form as it presents not only the design patterns created by the company but also the practical conclusions regarding their impact on the quality of the produced software and how easily one can modify it during the implementation or update. The paper underscores the fact that tests should be based on knowledge and experience in order to check the functionality as well as reliability, scalability, and effectiveness of the system for its users, clients, and administrators.

The development of good design patterns and making them a part of the company's policy can allow for the standardization of the structure of the process of software production (and supporting tools). The standardization allows for the control over the project and for the timely reaction for the updates (change management), as one of its principles is repeatability with respect to the procedures regarding the risk assessment and reduction, information management, tasking, methodology of testing, reporting, preparation and circulation of documents.

The tasks and aims in the IT project require proper methods and keys to be fulfilled. The design patterns regarding the design and implementation facilitate

that. But they are also a challenge for the IT specialists who must gather the related knowledge, experience and good practices and put it within the more systematic framework.

REFERENCES

[1] Andrew J. K., Abraham R., Beckwith L., Blackwell A., Burnett M., Erwig M., Scaffidi C., Lawrance J., Lieberman H., Myers B., Rosson M. B., Rothermel G., Shaw M., Wiedenbeck S. (2011) *The state of the art in end-user software engineering*

[2] Beck J., Almstrum V. L., Ellis H. J.C., Towhidnejad M. (2009) *Best practices in software engineering project class management*, In Proceedings of the 40th ACM technical symposium on Computer science education (SIGCSE '09)

[3] Bruegge B. D, Allen H. (2009) *Object-Oriented Software Engineering Using UML, Patterns, and Java,* Prentice Hall Press

[4] Fowler M. (2004) *UML Distilled: A Brief Guide To The Standard Object Modeling Language*, Pearson Education Inc.

[5] Graham I. (2004) *Inżynieria oprogramowania – metody obiektowe w teorii i praktyce*, WNT, Warszawa

[6] Heckman S., King J., Winters M. (2015) *Automating Software Engineering Best Practices Using an Open Source Continuous Integration Framework*, In Proceedings of the 46th ACM Technical Symp. on Computer Science Education (SIGCSE '15)

[7] Hudepohl J., Dubey A., Moisy S, Thompson J., Niederer H. M. (2014) *Deploying an online software engineering education program in a globally distributed organization*, In Companion Proceedings of the 36th International Conference on Software Engineering (ICSE Companion 2014)

[8] Larman C., Vodde B (2008) *Scaling Lean & Agile Development: Thinking and Organizational Tools for Large-Scale Scrum* (1 ed.), Addison-Wesley Professional

[9] Larman C. (2004) *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition)*, Prentice Hall PTR, Upper Saddle River, NJ, USA

[10] Lennart C.L. K., Vermaas R., Visser E. (2011) *Integrated language definition testing: enabling test-driven language development*, In Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications (OOPSLA '11)

[11] Rodrigues C., Neto L., Santana de Almeida E. (2012) *Five years of lessons learned from the software engineering course: adapting best practices for distributed software development*, In Proceedings of the Second International Workshop on Collaborative Teaching of Globally Distributed Software Development (CTGDSD '12)

[12] Szejko S. (2002) *Metody wytwarzania oprogramowania*, MIKOM, Warszawa