International Journal of Industrial Engineering and Management (IJIEM), Vol.1 No 4, 2010, pp. 163 - 172
Available online at www.ftn.uns.ac.rs/ijiem
ISSN 2217-2661

A Comparative Overview of the Evolution of Software Development Models

Predrag Matković
University of Novi Sad - Faculty of Economics, Segedinski put 9-11, 24000 Subotica, Serbia, pedja_m@ef.uns.ac.rs
Pere Tumbas
University of Novi Sad - Faculty of Economics, Segedinski put 9-11, 24000 Subotica, Serbia, ptumbas@ef.uns.ac.rs

Received (09.12.2010); Revised (24.12.2010); Accepted (24.12.2010)

Abstract

Information system (IS) development began as early as 1940. Up to the 1960s, IS development was based on IT pioneers' individual knowledge, so that this period is referred to as pioneer era, and some sources even use the term heroic age. The emergence of software crisis also saw the first forms of organised and systematised software development. Systematizing and organising the development efforts was concretised through a multitude of methodologies which appeared in the meantime, and were used for determining the structure, as well as planning and monitoring development procedures.
Academic and professional literature offers a multitude of definitions of the term methodology. Most definitions explain this term as a set of clear rules and principles for the application of methods and/or procedures, with the aim of resolving various problems. More specifically, in the software development area, methodologies can be viewed as a set of rules and principles for using methods and/or procedures applied when conducting development activities by pre-defined roles in the software development process, with the use of input artifacts, aimed at obtaining output artifacts as the constituent elements of the future software. Methodologies are built on theoretical foundations, and are all based on one or more development models.
In this paper, the term model refers to a systematic and logical description of a development process, without detailed elaboration of activities, performers, inputs and outputs. The paper provides an evolutional overview of development models, with their comparisons, and considers the expected avenues of future development.

Key words: *software development models, evolution, comparision*

1. INTRODUCTION

The most significant phenomenon in the systematization and organisation of the software development process *Systems Development Life Cycle* or *Software Development Life Cycle* (SDLC) (Fig. 1).

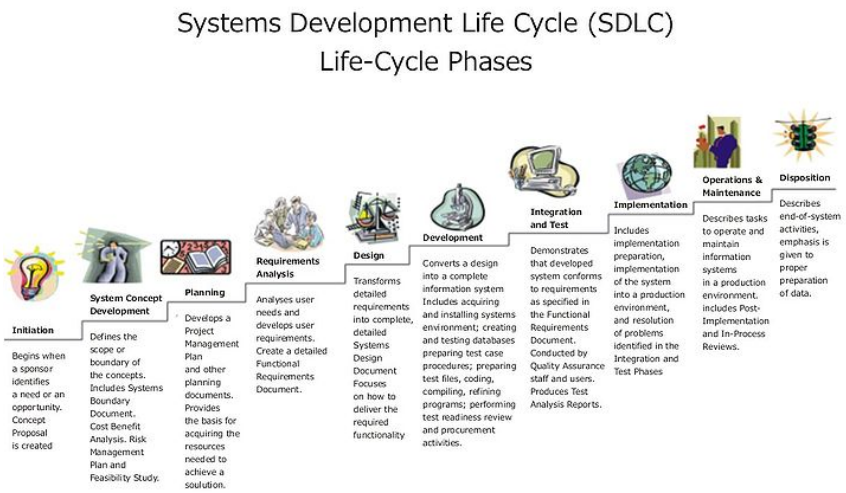

Figure 1. System development life cycle [1]

SDLS is a process of creating and adapting software products, as well as a basis for creating methodologies and models in software engineering. Since the emergence of SDLC in the early 1960s to date, all development models and all methodological approaches have incorporated elements of SDLC in their content [3].

Models and methodologies were developed both on strict adherence to SDLC on the one hand, and its critique and searching new alternatives in software development on the other. Those 1960s principles have served as a foundation for the construction of a multitude of development models to date, which can all be structured into two core categories:

- models based on sequential approach and
- models based on iterative approach.

2. MODELS BASED ON SEQUENTIAL APPROACH

The most significant model based on sequential approach is the waterfall model which is, together with its modifications, one of the most common development models in software development history. The original model was developed as early as 1970, by Winston W. Royce. When describing his model, he himself did not use the word „waterfall", but, due to the manner of presenting the essence of software development (Fig. 2), the model was named so.

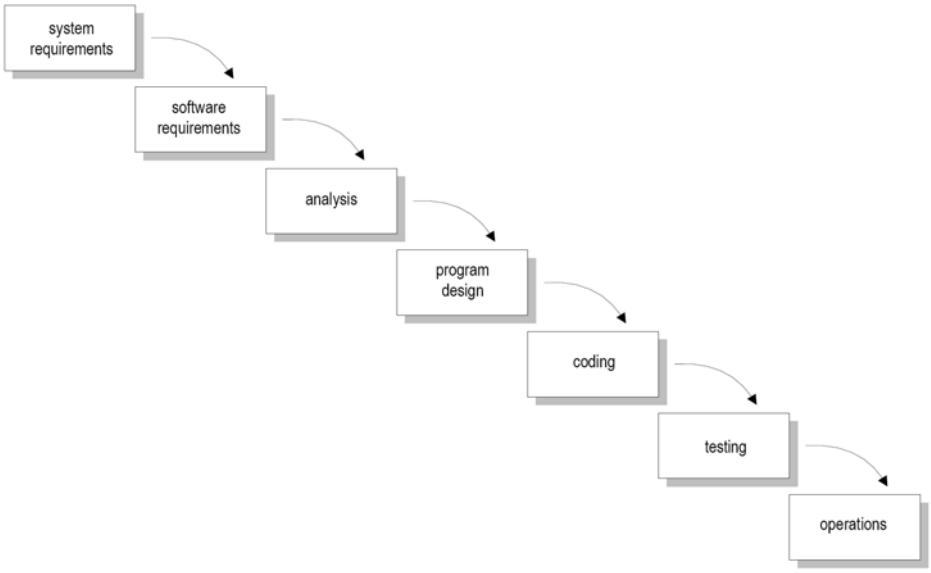

Figure 2. Waterfall model [2]

The key detail in Royce's model presentation is the positioning of development phases in the sequence of their execution. Supposing that the time in the model is distributed along a horizontal axis, it is obvious that the subsequent development phase follows only after the closure of the previous. This feature is the model's key characteristic, neglected in most of the later interpretations. Authors tended to neglect the time dimension and their graphic presentation included partially overlapping phases, thereby creating a misconception that development phases can flow simultaneously.

The waterfall model is based on SDLC. Initially presenting his model, Royce already expressed doubts as regards its successful practical implementation, so that he presented a theoretically ideal model plus its empirical equivalent (Fig. 3). The reason for such a model implementation lies in the fact that most verifiable outputs are available in the testing phase, when the procedure is, most often, returned to the software requirement phase, or, less frequently, to the design phase.

Figure 3. Royce's modifications of the original waterfall model [2]

"Royce further suggested that a 30-month project might have a 10-month pilot model and justified its necessity when the project contains novel elements and unknown factors (hardly a unique case). Thus, we see hints of iterative development, feedback, and adaptation in Royce's article. This iterative feedback-based step has been lost in most descriptions of this model, although it is clearly not classic iterative and incremental development." [11]

Although the shortcomings of the waterfall model were noticed even by its author, it became the basis for building a multitude of later models, and established itself as the most common model in software development history. The key advantages contributing to the model's popularity include:

- as a strictly defined model, it is characterised by standardised activities described in detail in all development phases;
- it includes testing, i.e. verification of completed operations and obtained results at the closure of each development phase;
- detailed and high-quality documentation is generated in all development phases, simultaneously with the execution of individual activities; and
- individual participants in the development process are comparatively easy to replace.

The model's disadvantages, which initiated its numerous modifications, are:

- inflexibility in the division of the development activities into separate phases and lack of back circuit i.e. feedback between stages;
- errors not removed in individual development phases, during product testing or verification, can have tremendous distortion impact on the development as a whole;
- impossibility of iterations during the realisation of development, as these cause serious problems and confusion in model implementation;
- difficulty of adaptation to uncertainty which tends to exist at the start of the project, when it is very hard for the users to explicitly specify all their software requirements;
- lengthy development process, so that users have to be very patient and persistent, because the working versions of the software are unavailable before the end of the development activity, until when there is only a written specification of the future software's functionalities;
- only a fully completed product is usable by the users; and
- high development costs.

One of the most interesting comparisons of the waterfall model with real-life phenomena, depicting its shortcomings, is Steve McConnell's comparison of the waterfall model and the lifecycle of salmon. The key objections to the waterfall model stated in this comparison can be summarised as: difficulties in encompassing the complete problem at the beginning of the development; difficulties in returning to previous development phase; and the fact that it is impossible to precisely separate activities, techniques and tool utilisation by phases. The author of the critical view of the waterfall model claimed that "it is possible to swim upstream, but the strain is so intensive that it can kill". This statement is graphically represented in Fig. 4. Various versions of the modified waterfall model have been developed in order to mitigate the disadvantages of the waterfall model.

Figure 4. Salmon life cycle as the critique of the waterfall model [4]

The so-called Sashimi model was developed by Peter DeGrace, as a modified version of the waterfall model (Fig. 5). The key feature of the Sashimi model is the possibility of overlapping development phases, i.e. introducing feedback into the classical waterfall model. The idea on which the model is based is identification of errors made on time, while the development phase is still in progress. For instance, errors made in the design phase are identified during implementation, while design is still in progress.
Another important feature of the Sashimi model is different treatment of documentation. Whereas the documentation in the classical waterfall model is exchanged by teams in charge of completing individual phases, Sashimi model treats the documentation as a unified document. Such an approach results in a significant reduction in the documentation volume.
The approach to completing the development process modified by the Sashimi model also has its shortcomings. These are, above all, unclear key development milestones, difficult monitoring of individual activities, and communication problems.

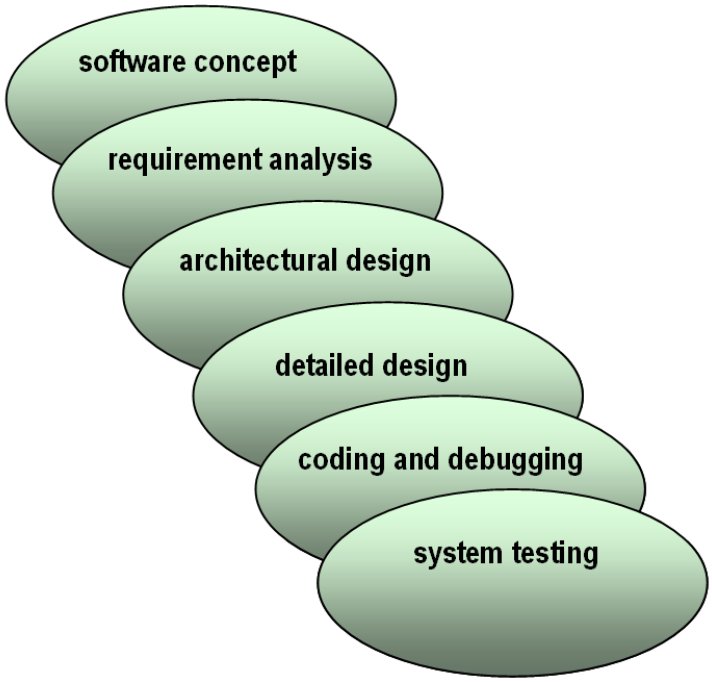

Figure 5. Sashimi waterfall model with overlapping phases [4]

In addition to the Sashimi model, literature also mentions waterfall model with subprojects and waterfall model with risk reduction. All the modified models share one feature, which is adding iterative elements into the classical sequential waterfall model approach, thereby representing their possible combination. Iterative elements, however, are added in the solution field, which remains the key disadvantage of all modifications.

3. MODELS BASED ON ITERATIVE APPROACH

Analysing academic and professional literature, it is easy to get a wrong impression that the use of iterative approach began with the emergence of object oriented development, and especially with agile development. Facts give quite the opposite evidence. In Royce's original article, describing this model, the author of the waterfall model argues that the classical sequential approach is not applicable to the entire project. He states that, if a project lasts for thirty months, a pilot version of the programme is released after ten months. After the user feedback, this version undergoes new iterations. So, in the text which represents the manifest of sequential approach, the author himself includes iterative elements.

3.1. The prototype model

The first generally accepted model based on iterative approach is the prototype model (Fig. 6). As the graphic representation of the model shows, it is a purely iterative approach. The first serious discussion on prototype-based software development was opened by Frederick Phillips Brooks Jr. A multitude of prototype types have appeared since then, all based on iterative approach.

The prototype-based development model begins by gathering and selecting the users' basic requirements. The development team and the users define the overall goals of software product development together, identifying all the known software product development objectives, and determining the areas requiring further activities of more precise definition. This is followed by "rapid" design, focussed on the realisation of the software aspects visible to the user (input and output formats, etc.). A prototype is developed following such design. Its purpose is to refine the requirements of the software developed. Refinement is iterative and

continues until the prototype meets the user's requirements, and, at the same time, enables the designer to fully understand the requirements to be met. Ideally, the prototype design serves for identifying user requirements. The subsequent design phase follows when the prototype is proved to be satisfactory by user evaluation, and so on until the final product for use is obtained [10].

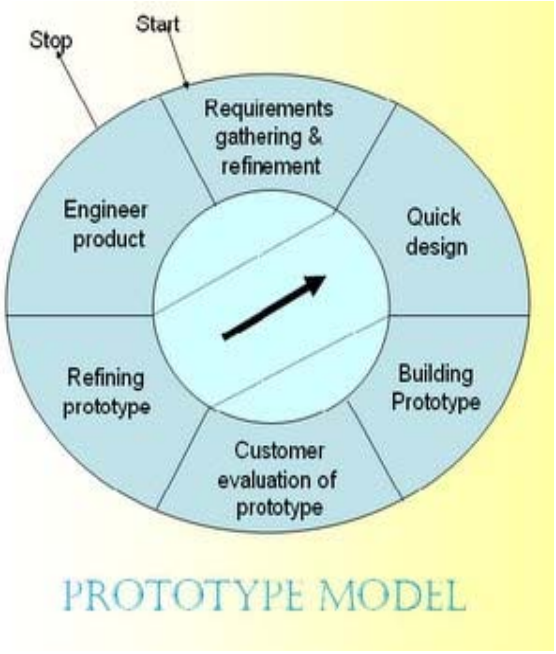

Figure 6. Prototype model [5]

The advantages of using the prototype model are:

- increased speed and creativity in development;
- constant provision of the product's working versions, which can serve for analysing functionality, performance, adaptability and development costs;
- users are fully involved in development and can constantly change their requirements, which enhances the quality of the total process.

This model also has its disadvantages, i.e. the application of the prototype model can be questionable for several reasons:

- resources cannot be accurately assessed and planned with high quality;
- the user observes the product's working version not knowing how the software segments are interconnected, or that the quality or maintenance were considered over a longer time period due to the implementation speed;
- the developer often resorts to compromise at the expense of software quality, with the aim to get the built prototype functioning, so that later on, such less ideal solutions, or better to say lower-quality solutions remain as an integral part of the final product;
- high probability of failure in replacing the prototype with the final solution; and
- in most cases, documentation created in the process of developing the final solution tends to remain uncompleted.

3.2 The spiral model

The model that made the greatest contribution to promoting iteration in software development is the model shown in Fig. 7. The spiral model was presented in 1988 by Barry W. Boehm. Viewing the appearance of the model, out of the context of time, as well as contemporary technical and technological situation in software industry, could lead to wrong conclusions and influence the comprehension of its significance.

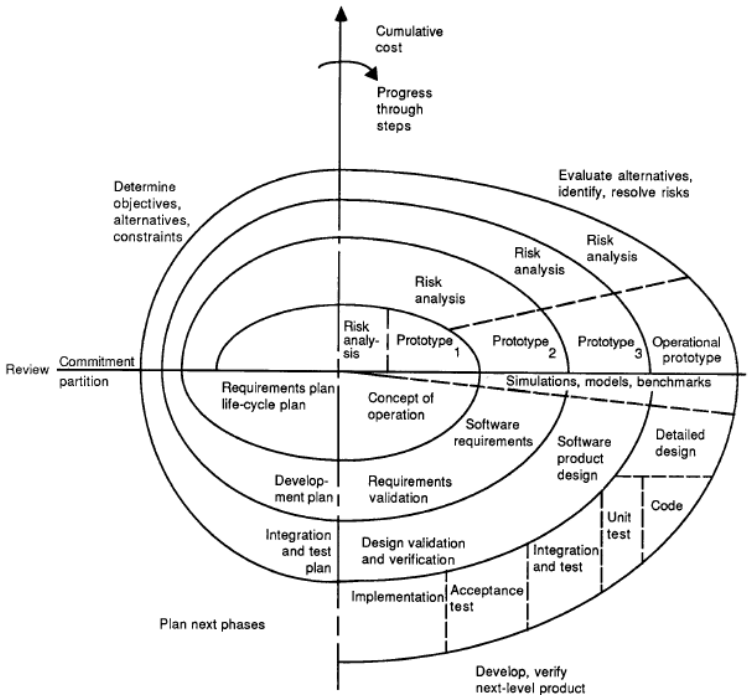

Figure 7. Spiral model [6]

The spiral model was devised as an ode to iteration. As an introduction to his work, Boehm quotes the following statements:

- „Stop the life cycle – I want to get off!“
- „Life-cycle Concept Considered Harmful.“
- „The waterfall model is dead. No it isn't, but is should be.”

These are only some of the quotations used by Boehm to start the theoretical debate on the model. The late 1980s was a period when the throne of structure-based development started shaking, and object-oriented development started taking up its place as the dominant software development method. Object-oriented programming languages such as ADA, SmallTalk and C++ had already fully demonstrated their strength. It had already become clear to everyone that structural programming techniques in software implementation were being abandoned. What was missing was coordination of other development phases. This was not brought on by the spiral model, but it did initiate the theoretical revolution of dethroning structure-based development. This is why Boehm and his model take up such a significant place in the history of software engineering.

The essence of the spiral model is risk assessment, used as a basis for decisions on further software development. As the output result, the end of each spiral produces a prototype which is the object of evaluation. The model also provides activities conducted along each spiral and in each of its segment (the spiral is divided in four with a coordinate system). The output points of each development phase are strictly determined, as inherited from the waterfall model, and developed prototypes evaluated are the inheritance of the prototype model. The defined development activities are clearly distributed along the spiral [9].

The model's fundamental tenet is that a given sequence of activities is iterated in software development and maintenance. Activities are initially performed with a high degree of abstraction, and then gradually move to details.

The advantages of the use of the spiral model are:

- production of functional software in a short time period;
- flexibility in the engineering phase and the possibility of combining various approaches to software development;
- possibility of risk assessment at any time and abstraction level, thus providing for timely response to observed risks and a mechanism for their reduction by the application of prototype model; and
- the model retains the systematic approach taken over from the waterfall model, with the possibility of iteration.

The spiral model's disadvantages are:

- absence of correlation with the existing standards, i.e. lack of standards for this software development model;
- the model requires more uniformity and consistency in development;
- the model is comparatively capital-intensive for application in small-scale product development, for risk analyses require specific expertise;
- great problems are created in situations when risks are not detected in due time or at all, thus producing multiplicator effect in its development.

3.3. The Unified Process model

The principles of the spiral model made a significant contribution to the development of other models, with a higher level of practical implementation. However, there remains another problem, which none of the models took into consideration before the mid-1990s. It is the treatment of the business problem as a whole, with decomposition of the entire system in the domain of solution. All of the concepts of the object-oriented approach practically craved for decomposition of the problem into constituents, but this had never been considered.

The mid-1990s finally saw the emergence of theoretical considerations that started applying iteration much earlier in the development process. Instead of previous segmentation of the solution, the focus was transferred to segmentation of the problem. The time dimension was distributed in relation to the content dimension (Fig. 8). Thus the development process was divided into multiple iterations, which were carried out based on small sequence approach. The key risk prevention effect was achieved by significantly reducing the time gap between error occurrence and detection. In addition, the user is much more actively involved in the software development process, thereby additionally contributing to risk reduction. The model promoting such a way of thinking is the Unified Process model (Fig. 9).

The Unified Process model first appeared in academic and expert sources in a 1999 book by Ivar Jacobson, Grady Booch and James Rumbaugh. The term Unified Process originated from the Rational Unified Process model, IBM's trademarked brand. Ever since, this has been the most common software development model.

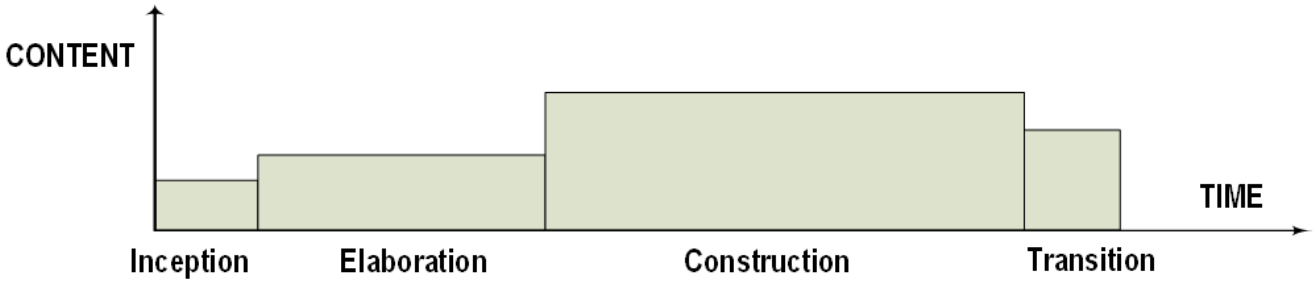

Figure 8. Unified Process model – Contents deployed from time stretch [7]

As a software development model, Unified Process recommends disciplines-based approach, with the aim of allocating tasks and responsibilities within the development team. The Unified Process software development is set between two dimensions, time and content. The time dimension is structured in four phases: inception, elaboration, construction and transition. The content dimension comprises six principal and three auxiliary disciplines. The principal disciplines are those of business modelling, requirements, analysis and design, implementation, testing and deployment. The auxiliary disciplines are configuration and change management, project management and environment. Each element of the process matrix is a combination of unified process static structure elements and their distribution over time.

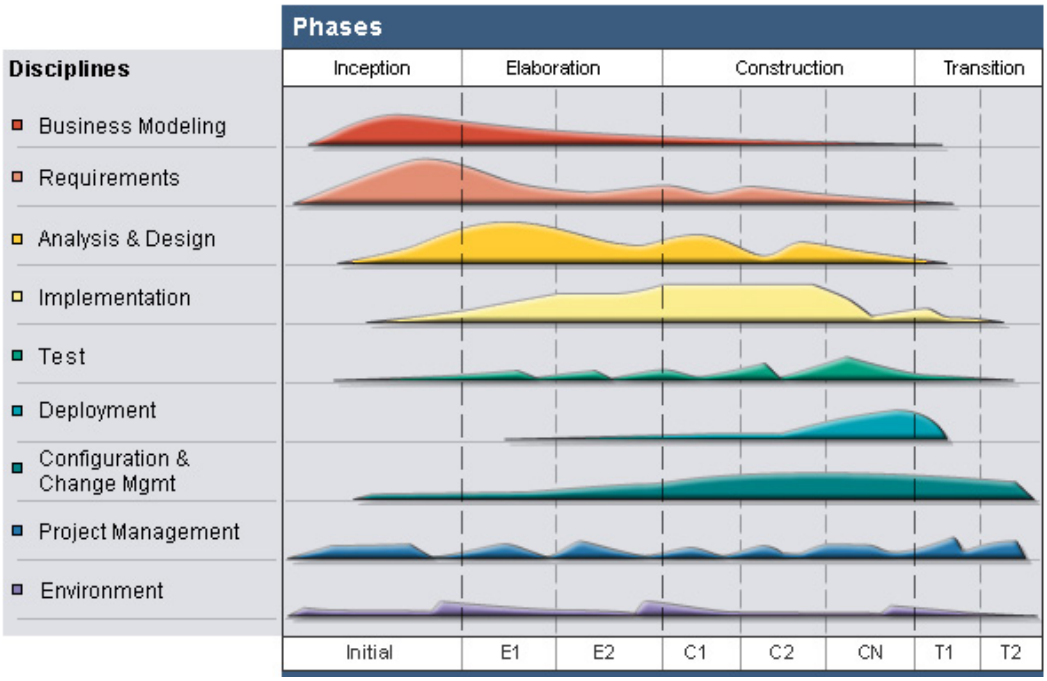

Figure 9. Unified Process model [7]

3.4. The Microsoft Solution Framework (MSF) model

Most of the significant development models are based on a combination of iterative and sequential approaches. Representing all those models is practically impossible, so that the following section of the article presents the MSF model, conceived by Microsoft's experts in order to set the theoretical basis for developing Microsoft products. It was practically unknown to the public until 2004, when it was presented in MSF documents published on Microsoft's pages.

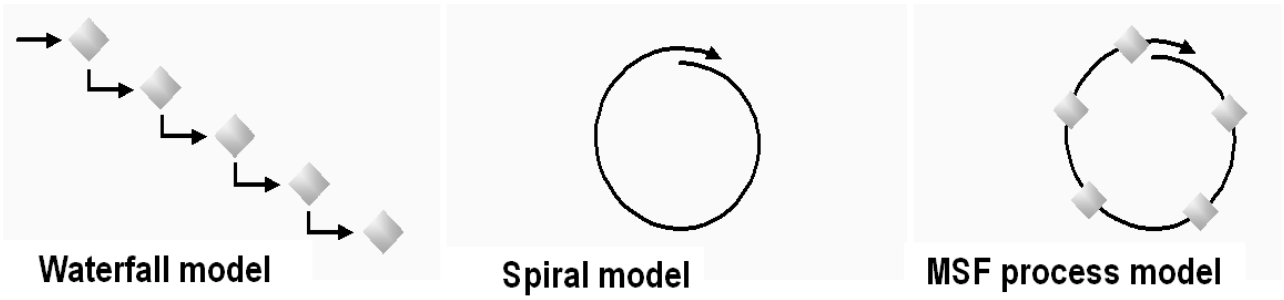

Figure 10. Simplified representation of the waterfall, spiral and MSF models [8]

The MSF model is based on the positive characteristics of the spiral and waterfall models. If simplified, these two models, as well as the newly built one, can be represented as in Fig. 10. The waterfall model comprises a sequence of milestones representing the closure of individual stages. Before moving to the subsequent development phase, all the activities from the previous phase must be completed. Such a development model is the best solution for projects with predefined and unchangeable requirements. The spiral model is based on continuous redefinition of user requirements and pre-calculations. This model can be highly effective when used for rapid application development and small projects. However, on larger projects, without clearly established milestones, software development process in this model becomes chaotic.

The MSF model has the following characteristics:

- phase- and milestone-based approach;
- included iteration; and
- integrated approach to solution building and installation.

The lifecycle of one project version is divided into five phases: envisioning, planning, developing, stabilizing and deploying (Fig.11). At the closure of each phase there is a principal milestone, and within it there are also internal milestones. What is completed in the previous phase is delivered at the principal milestones. Good communication and detailed information provided to users about all activities are major prerequisites for project success.

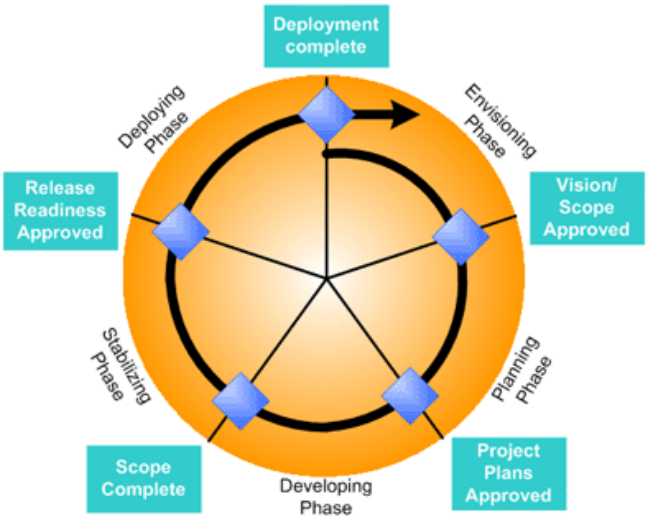

Figure 11. MSF model [8]

3.5. Agile models

Frequently exceeded deadlines and budgets in the realisation of software development projects, permanent growth in the complexity of technology and recurrent changes in user requirements have lead to the emergence of new development models towards the end of the 20th century, and the creation of agile models, much more flexible and adaptable to change by their features, enabling users to actively participate in all the development phases and activities.

The agile approaches partially faced up to the core problem of the contemporary and also rapid software development. The prevalent idea is that teams can be more efficient in change management if they are capable of reducing the time and costs of exchanging information between participants in development in such a way as to shorten the time period between the moment the decision was made and feedback on the impact of this decision.

The starting tenets of agile models were that turbulent corporate and technological environments require a software development processes which not only creates changes, but also responds to them, and, at the same time, a process including responsible participants and their good organisations. Special consideration is given

to participants, i.e. their talent, skills and abilities, as well as their mutual communication. Participant-centric approach is the key feature of agile models, as the complete development process is adapted to individuals. [12]

In agile development teams, individual competencies represent the critical factor of project success. According to agile models, if the individuals involved in the project possess adequate qualities, they can attain the desired goal with any development process. If not so, there is no development process that can substitute their incompetence. At the same time, the lack of user support may easily destroy the development process, just like inadequate support may prevent project completion.

Agile processes highlight the unique abilities of individuals and teams. That is to say, processes cannot compensate for inadequate individual competencies. The teams are self-organised, with intensive communication within and outside organisational frameworks. These teams can change their structure at any point, so as to adapt to change. Agility implies that the team shares a common objective, mutual trust and appreciation, joint and swift decision-making process and the ability to deal with all ambiguities. An agile team working within a rigid organisation is bound to have difficulties, just like any individual working in an inflexible team. Cooperation among all management levels dominates in these teams, and in decision-making process; what is important is not the decision-maker, but cooperation and providing information for decision-making. A development process involves people of various skills, talents and abilities, working in a close physical environment and adhering the organisational culture. Individuals, environment and culture are tightly interdependent. Teams of up to nine members tend to achieve the highest success rate in agile development.

Agile development has proven to be successful in specific, complex, and change-intensive projects. An environment where this approach yields the best results is people-centred and cooperation-centred organisation culture. There is a multitude of developed agile development models, including Extreme Programming (XP), Adaptive Software Development (ASD), Dynamic Systems Development Method (DSDM), Scrum, Feature Driven Development (FDD), Agile Modelling (AM) etc. All of these are based on agile principles, but, at the same time, highly mutually different when it comes to the manner of realising the software development process.

4. CONCLUSION

Any development project, including software development projects, is based on the “holy trinity” of successful project implementation, meaning: satisfying user requirements, carrying out the project within the allocated budget and planned time. If any of these requirements is not met, the project is not considered to be completed successfully. The evolution of development model has been progressing, and is likely to keep progressing, towards diminishing project failure risks and minimising project maintenance costs. Historically, viewed through development models, the characteristic of progress has always been in direct correlation with reducing the risk of software error occurrence, cutting error removal costs, and cutting software maintenance costs.

The first attempts, based on SDLC, were reduced to the need for establishing order in the software development process, establishing the order of activities and output results, and placing these in the function of the subsequent development phases. This was a way to significantly reduce error occurrence risk. Establishing order certainly also contributed to cost-cutting in error elimination, as well as maintenance cost reduction. Technical and technological process and growth in expectations materialised in new types of user requirements opened new requirements and initiated further evolution of development models. The solution was sought in introducing a general iterative principle, which resulted in raised users’ importance and their involvement in the development process. Iterativity in all development phases is achieved by deploying the content of the development project into another dimension in relation to time, and establishing iterations as early as the problem domain. In addition to the overall iterative principle, the latter changes also included dynamism into the development process, considering changes in requirements occurring both in the development and the maintenance phase. For the purpose of achieving dynamics, project hierarchy was significantly reduced, and the concept of self-organising teams was introduced.

Further avenues of model development should be sought in primarily in meeting user requirements initiated by technical and technological progress. Due to corporate dynamics, the significance of users in the development process will keep growing to reach the status of users as full and equal members of development team. Adherence to the iterative principle will lead to further atomisation of the problem, which is already seen through the web service based development principles. Future development model solutions should be sought in optimum combination of general development principles established in the already existing models, in accordance with the newly-arisen environment requirements. As it can be seen from the text above, the development has had a predominantly evolutional character so far. Such a development trend is to be expected in future as well, meaning that future solutions will still be based solely on past achievements.

5. REFERENCES

[1] http://www.justice.gov/jmd/irm/lifecycle/ch1.htm (accessed: 12.July 2010)
[2] Royce, W. W. (1970), “Managing the Development of Large Software Systems: Concepts and Techniques” In: Technical Papers of Western Electronic Show and Convention (WesCon), Los Angeles, USA.
[3] Rama Mohan Reddy A., Govindarajuku P. and Naidu M. M. (2007), “A Process Model for Software Architecture”, IJCSNS

International Journal of Computer Science and Network Security, Vol.7, No. 4, pp. 272-280.
[4] McConnell, S. (1995), Rapid Development, Microsoft Press, Redmond, Washington, USA.
[5] Pressman, R. (1994), Software Engineering, McGraw Hill, Berkshire, England.
[6] Boehm, B. W. (1988), "A Spiral Model of Software Development and Enhancement", IEEE Computer, Vol. 21, No. 5, pp. 61-72.
[7] Rational Team (2005), Rational Unified Process V7.0, IBM Rational.
[8] Microsoft Team (2004), Microsoft Solution Framework White Papers, Microsoft.
[9] Pressman, R. (2005) Software Engineering,McGraw Hill, Berkshire, England.
[10] Sommerville, I. (2007), Software Engineering, Pearson, Essex, England.
[11] Craig Larman and Victor Basili R. (2003), “Iterative and Incremental Development: A Brief History”, IEEE Computer, Vol. 36, No. 6, pp. 47-56.
[12] http://www.ambysoft.com/unifiedprocess/agileUP.html (accessed: 12.November 2010)