American Journal of Computer Science and Technology
2023; 6(1): 1-9
http://www.sciencepublishinggroup.com/j/ajcst
doi: 10.11648/j.ajcst.20230601.11
ISSN: 2640-0111 (Print); ISSN: 2640-012X (Online)

Research on Contemporary Software Development Life Cycle Models

Liu Yuge, Tuyatsetseg Badarch*

School of Information Technology and Design, Mongolian National University, Ulaanbaatar, Mongolia

Email address:
ba.tuyatsetseg@mnun.edu.mn (Tuyatsetseg Badarch)
*Corresponding author

To cite this article:
Liu Yuge, Tuyatsetseg Badarch. Research on Contemporary Software Development Life Cycle Models. *American Journal of Computer Science and Technology*. Special Issue: *Advances in Computer Science and Future Technology*. Vol. 6, No. 1, 2023, pp. 1-9.
doi: 10.11648/j.ajcst.20230601.11

Received: January 12, 2023; **Accepted**: February 20, 2023; **Published**: March 4, 2023

Abstract: The rapid development of computer software and modern technology is perfectly integrated and has a huge contribution to the development of society. The paper covers the research of software life cycles of contemporary software systems, especially, we focus on their development stages, and models for their life cycles. We studied there are powerful and principal software systems such as system software, application software, and support software. Software is mainly divided into seven categories such as system software, application software, engineering/scientific software, embedded software, product line software, web applications, and artificial intelligence software depending on their usage. In addition, we analyzed the models including workflow, data flow, and action flow models. The software process model is expressed as a simplified abstract representation and a framework of the software development process including the various activities that make up the software process, software artifacts, and the actors involved in the development. The paper focuses on the life cycle of software that can be divided into six parts: planning, analysis of requirements, design and coding, testing, and operation and maintenance. In addition, we describe the relationship between the various parts of the software development process which is an iterative process with some feedback. We show development methods and tools to improve the efficiency and capability of software development. This paper presents potential software development life cycles described by the cycles from product development to end-of-life with phases such as problem definition, requirement analysis, system design, coding, debugging and testing, acceptance and operation, maintenance and upgrade to disposal. When planning long life cycles, we emphasize performing a detailed requirement analysis of the various functions that must be implemented in the software development stages. As a result of studies, we want to emphasize component assembly-based software engineering methods that have emerged in recent years, and these models have been widely used. Our study emphasizes the waterfall model which has been playing an important role as one of the earliest software life cycle models. The model describes some basic process activities of software life. At our study point, one thing became clear the software architecture has become obsolete and should be discarded to meet the system design requirements of a new software architecture to replace it. This paper presents that the selection of the software lifecycle model plays a crucial role in the software.

Keywords: Computer, Software, Life, Cycle, Models, Development, Application, Testing

1. Introduction

As computer technology continues to be applied to various fields, it profoundly affects people's lives and development and even civilization. Computer software engineering has adapted to the needs of the times, brought great convenience to various industries and fields, and created more conditions for the overall development of the industry. Computer software and modern technology are perfectly integrated and have a huge contribution to the development of society [1]. Analyzed from the point of view of computer definition, software is the application software developed through computer software technology, which can also be described as a computer system, and it is one of the results of the development of computer software

technology [2]. With the continuous innovation and development of computer software technology, the practical application of many software technologies tends to develop in the direction of high-end, simple and intelligent, which can provide more convenience to people's lives while driving, integrating and developing related industrial chains [3] and development [3]. The development of computers now tends to be multi-faceted, more modular and functional, and the steps required to develop a software are divided into many positions, which also aims to prevent the generation of software crisis, so there are still many opportunities for the development of contemporary students, and the prospects are also very broad [4]. As the needs of users are becoming clearer, the corresponding software technologies are keeping up with the contemporary trends and keeping up with the times, and computer software is constantly winning and losing. Research and development personnel are required to actively make strategies to enhance the technology so that the quality of software use can be improved more quickly to meet the differentiated needs of different inches of users. More importantly, it is important to make timely corrections to improve software vulnerabilities through continuous feedback from users [5]. This highlights the importance of the software life cycle in particular. The software lifecycle approach is applicable to the development of large-scale complex systems and is relatively widely used [6]. The life cycle of a software can be divided into six parts, namely, planning, analysis of requirements, design and coding, testing, and operation and maintenance. However, the relationship between the various parts of the software development process is not fixed, but it is an iterative process with some feedback, and in software engineering, this complex process can be specifically described and represented by a software development model. A software development model is a model that provides a structural framework for all the work and tasks related to the development, operation and maintenance of the system across the entire software life cycle, and gives the relationship between the various parts of the specific software development activities. Such as the waterfall model, transformation model and agile model [7]. The features and advantages of the life cycle can be fully highlighted in the development of complex industrial software to improve the global and holistic nature of the software development system [8]. The selection of the software lifecycle model plays a crucial role in the extensibility and plasticity of the software. Therefore, the problems and advantages of life cycle need to be studied.

2. Classification of Software Systems

Regarding the study, there are still many opportunities for the development of contemporary software systems, and the prospects are also very broad [4]. Software is mainly divided into seven categories such as system software, application software, engineering/scientific software, embedded software, product line software, web applications, and artificial intelligence software depend on their usages. In this paper, we focus on the main types of software systems of computers including system software, application software, and support software.

2.1. System Software

The system software is classified into four types including operating system software, computer language translation system software, system support, and service oriented software.

The operating system software controls and manages the computer whole system, it can be main interface between the user and the hardware system to keep two directional access.

The language translation systems are written in various programming languages, such as assembly language, C, Java and other high-level languages, however these never directly executed by the computer as source programs, and they must be translated, which requires language translation systems.

System support and service programs are named as tool programs. There are system diagnostic programs, debugging programs, scheduling programs for error fixing, editing, virus checking of programs that are configured to maintain the proper operation of a computer system or to support system development.

Database management systems are typical system software for creating and managing databases helping end users to create, protect, read, update and delete data in a database. The system software helps to serve as an interface between databases and end users or application programs. The system software ensures that data is consistently organized and remains easily accessible.

2.2. Application Software

The application software is designed to perform specific functions for a user. According to its service objects, the software is generally classified into general-purpose application software and special-purpose application software. For instance, the sole purpose of general software is described by the public application programs such as Microsoft word and excel, as well as popular web browsers like Firefox and Google Chrome, database, as well as public aided design and manufacturing CAD & CAM, computer networking. It also can encompass the mobile apps includes apps like Whatsapp, Wechat, Facebook, Candy Crush Saga, so on.

Dedicated application software is designed for a small number of users with a single objective. These are dedicated software for automatic control software for a machine tool equipment, data acquisition and data processing for a certain experiment, and supplementary teaching software for learning a certain course etc,. Regarding our research result, it is identified that principal problems in application software are mainly described by their maintenance based on user knowledge, programmer effectiveness, product quality, programmer time availability, machine requirements, and system reliability. Among of them, user knowledge

accociated problems account for huge amount of problems, as well as problems of programmer effectiveness and product quality are greater for older and larger systems [4, 5].

2.3. Support Software

Support software updates, upgrades, patches, fixes or supports new versions to assist software development, operation, and maintenance. Traditional support software were tools such as modeling tools, language tools, development tools, testing tools, and version maintenance tools, however, with the development of network technology, the demand for support software has promoted the development, deployment, operation, integration, management, security and maintenance of various network applications.

3. Role of the Software Models

Before defining the software lifecycle, it is important to define the software model. An abstract expression of a real system that answers the desired research question can be called a model. In general, a model is described as an abstract expression of a real software system and its problems. The problem is a domain by extracting the main factors and major conflicts to be understood and solved, However, the problem of a model can ignore minor factors that do not affect the basic nature. Models are extracted from real systems and, conversely, can be applied to other real systems by understanding them.

Models can be represented mostly in the form of common mathematical expressions, physical models, or graphical textual descriptions.

In this paper, it is identified the software life cycle model determines the sequence of the software development activities.

An appropriate software life cycle model should be chosen at the early stage of software development in order to simplify the software development process and reduce the difficulty of software development. Software life cycle models can help developers to improve software development efficiency, improve software quality, reduce software development costs, better monitor and control the software development process, and reduce risks [9]. The software process model can be divided into the following three main types:

Workflow model: It describes the sequence of activities, inputs and outputs in the software process, and the interdependencies between various activities. It always covers the organizational control strategy of the activities in the software process.

Data flow model: This model describes the activities throughout the process of transforming software requirements into software products. The activities perform the function of transforming input artifacts into output artifacts. The model clearly determines the transformation relationship of artifacts in the software process. The specific implementation measures of artifact transformation are not defined.

Role/action model: The model describes the different roles and actions involved in the software process. Especially, the actions accomplish the software process. Therefore, actions define the roles in the software process, the collaborative relationship between the roles, and the specific determination of the roles' responsibilities and activities.

These three types of software process models can in turn be specified in the specific software development process as software life cycle models. According to the time of development, in this paper, we focus on software life cycle models that are divided into traditional software life cycle models and modern software life cycle models.

4. On Software Development Life Cycle

The Software Development Life Cycle (SDLC) is a structured process for building high-quality and low-cost software. The SDLC originally consisted of five stages: planning, creating, developing, testing, and deploying. However, with the development of the software technologies, currently, SDLC is described by seven stages including planning, requirement analysis, design, development, testing, operation, and maintenance.

4.1. Planning Stage of Software Life Cycle

This stage describes the general objectives of the software system to be developed, giving its requirements and technical limitations in terms of functionality, performance, reliability, and interfaces. For the planning stage, the system analyst and the user cooperate with each other to study the feasibility of completing the task of the software system. In addition, they also explore possible solutions to the problem and estimate the available resources including computer hardware, software, human resources, costs, and benefits to be achieved. The planning stage includes the development schedule and planning for completing the development task. The final approval is determined by review from committee members. There are three main parts in the planning stage of the software life cycle including problem identification, the feasibility of the system development, and resources.

Problem definition: Through research, the team clarifies the problems to be solved, process objectives, and scale, then they form a preliminary requirement report for users and get confirmation from users.

Feasibility demonstration: Based on the preliminary user requirement, the planning report should be confirmed by the user and the real environment situations. In this study, we present that the feasibility of the software system is described based on requirements from technical, economic, and social aspects. We also emphasize that the selection of the requirements determines the right solution and form feasibility requirements.

Develop a preliminary project development plan: it includes a selection of resources, a definition of tasks, risk analysis, cost estimation, cost-benefit analysis, and project schedule.

4.2. Requirements Stage of Software Life Cycle

After determining that software development is feasible based on the requirements, a detailed analysis of the various functions to be implemented in the software is performed.

We say the requirements analysis phase is the most important stage, then this result influences to the success of the entire software development project.

In fact, the requirement stage always is determined by a change throughout the software development process, so a requirements change plan must be developed to cope with such changes and to protect the smooth running of the project. For the appropriate change influencing to the requirement and quality of the project, the following types of test help to determine the right solution.

At this stage, the requirements survey should be done. This survey is a detailed survey of the needs of the software and its use environment to grasp the requirements of users and to determine the working conditions of the software life cycle.

According to the situation, we analyze and study the function, performance, and environmental constraints analysis including function (i.e., what the system must do), performance (including software security, reliability, maintainability, accuracy, error handling, adaptability and user training) and environmental constraints (meaning that the software system to be developed must meet the requirements of the operating environment) of the software system, and obtain a consistent understanding with the users.

In this stage, one more important thing is that to prepare software requirement specifications to write the functional requirements, performance requirements, interface requirements, design requirements, basic structure, and development standards.

In addition, based on the validated software development standards and acceptance principles, the developing software system validation test requirements and user manual outline are the documentation to guide/ help users to understand the software.

4.3. Design Stage of Software Life Cycle

The design stage designs the overall structure of the software system based on the functions determined in the requirements analysis phase divides the functional modules determines the implementation algorithm of each module and writes specific code to form a concrete design plan for the software [11]. The design phase starts with the software requirements specification. In this phase, the entire software system is designed based on system framework design and database design. Software design is generally classified into the general design and detailed design. A good software design can express a good software program writing performance.

In the book "Information Architecture, Beyond Web Design", Louis Rosenfeld mentions that information architecture is an essential element of design that allows information about software applications and websites to be more easily retrieved and understood by users [12].

For instance, in the design phase, the overall system structure may be described in this way based on a diagram expression (Figure 1).

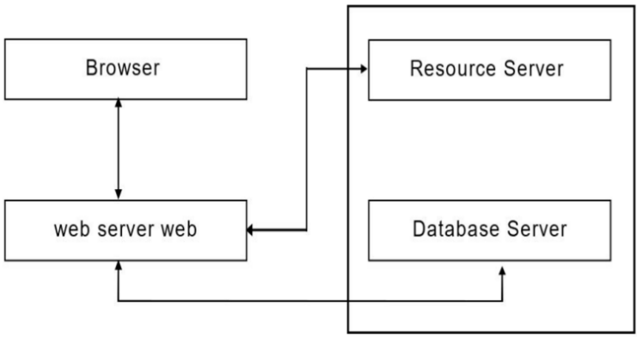

Figure 1. *System overall structure diagram.*

According to the software requirement specification, to establish the overall structure of the software system, the overall functions of the software system are classified into modules to form the functional structure diagram of the system.

Defining the interface of functional modules helps to perform the functions of modules and determine the relationship between modules, and give the definition of the interface of each module.

Designing a global database and data structure is one important part to define the properties of basic data items and data structures from the application problem domain, and designing the logical structure of the global database.

However, there are specific design constraints to define the boundaries of the software system and to give a description of the constraints on the system design [11].

At the manager level, preparing outline design documents is always to express the design stages documentation such as outline design specification, database or data structure specification, and assembly test plan.

In this study, we emphasize the detailed module design including the design of module functions, algorithms, data structures, and interface information between modules, and the development of module test plans.

Another important process of the stage is about preparing the detailed specification of the module that is summarized to form the detailed specification of the module.

4.4. Program Code Stage of Software Life Cycle

This stage is the conversion of the results of software design into computer-runnable program code.

To perform metrics on the software coding phase, people basically determine an evaluation baseline of what kind of code is of good quality and what kind of code is in need of improvement. It is first determined by the evaluation baseline by a true estimate of the software generic defect rate [13]. In this stage, uniform, standard-compliant writing specifications must be developed in program coding to ensure readability, and ease of maintenance, and to improve the efficiency of the program's operation. The programming stage is the

development main stage, and the management of this stage includes all organizational and decision-making activities from the specification phase to design, coding, and testing, and even the runtime phase, where the development manager schedules meetings, maintains contact with all participants and tracks development progress and work standards, gives guidance and makes decisions [14].

4.5. Software Testing

After the software design is completed, the progress of software development has to go through a rigorous design testing process to find out the problems in the whole design process and correct them.

The whole testing process is classified into three stages: unit testing, assembly testing (integration testing), and system testing. The testing methods are mainly white box testing and black box testing. In the testing process, a detailed test plan is planned and strictly follows the test plan in order to reduce the arbitrariness of testing [15].

Unit testing as module testing is to test the correctness of the smallest unit of the software design specification. The purpose of unit testing is to verify that the function unit correctly implements the functional, performance, and other design constraints in the software's detailed design documents, and to discover possible defects within the software unit. The types of tests for unit testing generally include documentation review, static analysis, code review, and dynamic testing [15].

After the unit testing is completed, the assembly testing process should be performed based on the basis of unit testing results. The assembly testing tests whether all software units are assembled into modules, subsystems, or systems in accordance with the requirements of the outline design specifications. The assembly test also determines whether each part of the work meets or achieves the corresponding technical specifications and requirements.

This is very important because, without the assembly test, we increase significantly the cost of software error correction.

In addition, we emphasize software system feasibility testing which actually tests the user's understanding of the system and its use effect, similar to the system operability test, which involves the system's function, the system's release, and the user's interaction effect with the system.

System feasibility testing mainly includes navigation testing, graphics testing, content testing, interface testing, etc. [15, 16]. Total testing of the software system is performed according to all functional and performance requirements defined in the software requirements specification and software validation testing guidelines [16].

4.6. Operation and Maintenance Stage of Software Life Cycle

In general, we can acknowledge that the Operation and Maintenance (OP) phase is the life cycle phase in which users operate the software and utilize the finalized solution of the software and services the product provides. After OM period, the programmers provide continuous maintenance and user support until the software is finally accepted by users, after which a maintenance organization or users become responsible for it. Because this process is a general process for all types of software [9].

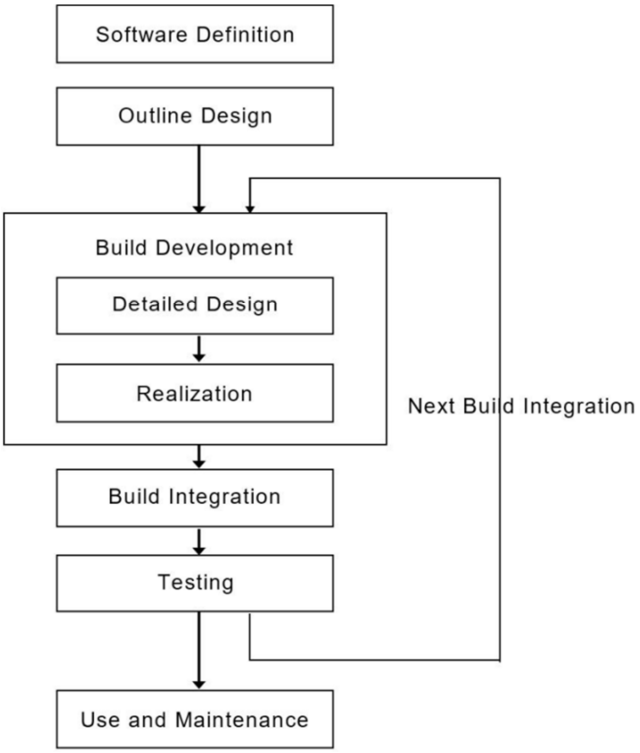

Figure 2. *Incremental model structure.*

During this phase, the software may need to be modified during operation for a variety of reasons. The reasons for this may include bugs or errors in the software during operation that needs to be corrected. In addition, these may include appropriate changes to accommodate changes in the software's operating, as well as changes to enhance the software's functionality. In this paper, we focus on the principal factors that may affect the length of the software life cycle.

In the operational and maintenance phase, we always need to consider the use phase, the maintenance phase, and decommissioning subphases. The use phase describes the installation time of the software in a user-defined operating environment for use. Maintenance phase help to make changes to the software product or respond to changes in software requirements, and write maintenance reports for all maintenance [13, 14].

However, when we consider the decommissioning phase, there are special time frames just started. For this, we studied the frame: the software has completed its mission, or a new software life cycle is almost to start, the original software product is terminated, or the software is discontinued. The integrated software life cycle process stages can be mapped out in a software life cycle mode.

5. On Architectural Design of the Life Cycle Models

In general, we all know that software architecture is described by the structures of a system that comprise software elements and the relationships among the elements.

5.1. Non-Formal Description of Software Architecture

Despite the common use of natural language, software architecture is creative and pioneering [18]. However, when a software architecture is created, its ideas are usually simple and often expressed by software designers in a non-formal natural language of concepts. For instance, client/server architecture is one type of software architecture that evolved from master-slave to accommodate the requirements of distributed systems.

5.2. Specification Description and Analysis of Software Architecture

The specification description and analysis phase of the software architecture is defined by applying a suitable formal mathematical theoretical model to the non-formal description of the architecture. In order to make the description of the software architecture accurate and ambiguity-free, we must analyze the nature of the software architecture, such as deadlock-free, security, and activity to obtain a perfect formal specification description. The analysis of the nature of the software architecture is beneficial for the selection of the appropriate software architecture at the time of system design, thus serving as a guide for the selection of the software architecture and avoiding blind selection [17].

5.3. Refinement of Software Architecture and Its Verification

The software architecture refinement and its verification phase to complete the design of the software architecture have been grabbed. The refinement of the software architecture of large systems is always through the abstraction to concrete, progressive refinement, and achievement, due to the complexity of the system, abstraction is an essential way of thinking when dealing with complex problems and objects, software architecture is no exception [16]. However, we learned that too much abstraction always makes the software architecture difficult to implement in the system design.

Hence, if we see that the software architecture abstraction granularity is too large, it is necessary to refine the architecture, refinement, until it can be implemented in the system design.

In each step of the software architecture refinement process, the need for different levels of abstraction of the software architecture antelope, to determine whether the more specific software architecture and the more abstract software architecture have semantic consistency, and can implement the abstract software architecture [17].

5.4. Implementation of Software Architecture

The implementation phase of the software architecture describes the refined software architecture that is penetrated into the system design. It naturally organizes the components and connectors of the software architecture to form the framework of the system design, therefore, the architecture can be implemented into the software design and construction.

5.5. Evolution and Extension of Software Architecture

After the implementation of the architecture, the evolution and extension phase of the software architecture starts. According to the requirements of the system, which is called the evolution of the software architecture, when implementing the software architecture, non-functional requirements such as performance, fault tolerance, security, interoperability, adaptivity, and others affect the extension and modification of the software architecture [18].

Since the evolution of software architecture is often caused by non-formal requirements description, the evolution and extension require repeating the first step.

In the case of functional and non-functional natures, the understanding of the software architecture has to be designed. Furthermore, reverse engineering and re-engineering of the software architecture are required.

5.6. Software Architecture Provision, Evaluation, and Metrics

The provisioning, evaluation, and metrics phase of the software architecture is carried out through the implementation of the software architecture in the system design. It is also performed by the actual operation of the system, the qualitative evaluation of the software architecture, the quantitative metrics for the reuse of the software architecture, and the acquisition of lessons learned.

5.7. Evolution and Modification of Software Architecture

If the software architecture has evolved and modified several times, the software architecture has become more difficult to understand. In this phase, maybe the software architecture can not meet the requirements of the system design, and can not adapt to the development of the system. At this point, the software architecture of the re-engineering project is neither necessary nor feasible.

Hence, it indicates the software architecture has become out of date when many modifications occurred without proper requirements.

6. Software Life Cycle Model

The software life cycle is the cycle from product development to end-of-life with phases such as problem definition, feasibility analysis, general description, system design, coding, debugging and testing, acceptance and operation, maintenance and upgrade to disposal [10].

These phases can be categorized into 3 periods, namely the software definition period, the software development period, and the software operation and maintenance period. Dividing the software life cycle into several phases, each phase will have a clear task, and then the software gradually completes the tasks in each phase. The process helps software development to manage and control complex processes easily.

6.1. Waterfall Model

Historically, the waterfall model has played an important role. The waterfall model is one of the earliest software life cycle models. The model describes some basic process activities of software life. The basic activities include planning, requirements analysis, software design, program writing, software testing, and operation and maintenance, and stipulates that they are from top to bottom [18, 20]. The fixed sequence of mutual convergence, like a waterfall, falls step by step as the waterfall model.

In the waterfall model, various software development activities are carried out in a linear manner. The current activity accepts the work results of the previous activity and implements the required work content. The work result of the current activity needs to be verified. If the verification is passed, the result is used as the input for the next activity, and the next activity is continued; otherwise, the revision is returned. The characteristics of the waterfall model are as follows sequence and dependence among the various stages, the dependence of the quality assurance mechanism as well as the principle of postponing realization [19, 20].

Hence, it is clear that the process of the waterfall model depends on the result of the work of the previous stage as an input of the work of the next stage. In other words, each stage is built on the correct result of the previous stage, and the mistakes and omissions of the previous stage will be hidden to the next stage. This kind of error can sometimes even be catastrophic [20]. Therefore, after each stage of work is completed, it must be reviewed and confirmed, which is very important. The waterfall model uses stage review and document control to ensure the progress and quality of the software project, but the model lacks the flexibility to adapt to changing needs, freezing of requirements, no backtracking and feedback loops, and is not suitable for object oriented projects.

6.2. The Incremental Model

The incremental model combines the basic components of the waterfall model and it has the iterative features of the prototype implementation. The model uses linear sequences that are staggered with the progress of the schedule. Each linear sequence produces a releasable "increment" of the software " [21]. When using an incremental model, the first one increment is often the core of the product. This process is repeated after each increment release until the final perfect product is produced. We studied that the incremental model combines the advantages of the waterfall model.

The incremental model does not deliver a complete runnable product at each stage, however, the entire product is broken down into several components, hence developers deliver the product component by component.

6.3. The Fountain Model

The model supports object-oriented development methods. The model provides support for software reuse and integration of multiple development activities in the software life cycle. When we use this model, the various stages of the software development process are overlapped and are repeated many times. Compared to the Waterfall model, The fountain model is suitable for developing Object Oriented projects [20].

The functional modules are not completed at one time but are like fountains. There can be a fallback into the previous phase providing an iterative fashion. The water can be sprayed up and down, either in the middle or to the bottom.

There is no specific sequence requirement for each development stage, the phases and sequence order remains the same for both Waterfall and Fountain model [20].

The fountain model allows for the overlap of activities between development stages including portfolio, maintenance, practice, design, and analysis stages which shows the activities cannot start before others (Figure 3).

The model has high reliability, reusability of code, maintainability, understandability, and robustness. In addition, we emphasize that the model has no freezing of requirements, a possibility of changes, and iterations. As a result of iterations, the model often repeats work many times, and related functions are added to the evolved system in each iteration.

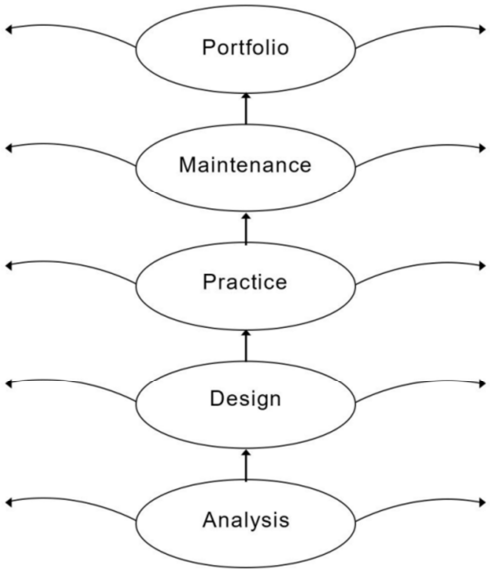

Figure 3. *Fountain model.*

6.4. The Spiral Model

Spiral Model using a cyclical approach used for large projects which involve continuous enhancements. systems

development. The main activities such as planning, risk analysis, engineering, and evaluation activity are performed in each iteration. Therefore, the main feature is that the spiral model has specific activities that are done in one spiral where the output is a small prototype of the software system. The types of processes are then repeated for all the spirals until the entire software system is built.

When we use the model, typically, project managers can demonstrate certain concepts to clients at an early stage. The model is centered on an evolutionary development method, using the waterfall model method at each project stage.

We say this model has a high performance to use for complex software systems where we can develop and deliver smaller prototypes and can enhance them to make complex software systems.

6.5. The V-shaped Model

The waterfall model treats testing as an independent stage after software implementation, making potential errors in the analysis and design stages [20].

The V-model is a type of SDLC model where the process executes in a sequential manner in V-shape that is also known as Verification and Validation model (Figure 4). The model has two main design and testing phases.

The design phase starts with the requirement analysis to communicate with the customer to understand their requirements and expectations. After this requirement, the system design, and the infrastructure setup are done for the developing purpose of the software project.

Within the design phase, the detailed architectural design is performed further into modules taking up different functionalities for the model design. The final stage of the design phase is building modules. The detailed design of modules is named as Low-Level Design. The figure shows this principle of the model. Once completing the design phase, in this model, we start the testing phases for all types of testing such as unit testing, integration testing, system testing, and user acceptance testing (Figure 4).

This model emphasizes the importance of testing. It closely links development activities with testing activities. Each step will be more complete than the previous stage.

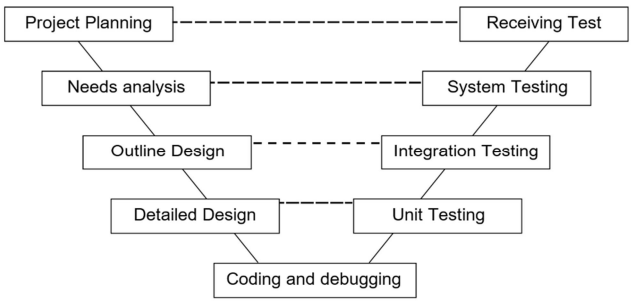

Figure 4. *V-shaped model.*

6.6. The Component Based Assembly Model

In general, we know there is some reused software in many software projects. When reused software is used then the design or code in one project is repeated in another project, and reuse occurs naturally. Similar to this reusable idea, we describe the component-based assembly model. In this model, classes work as reusable components.

The component-based assembly model always uses object-oriented technologies. In the technologies, classes are main the entities that encapsulate data and algorithms to build the application. Classes are reusable components.

As a result of the model, the reuse of components improves the reliability and ease of maintenance of the software, and the program has fewer effects when it is modified.

During a system development process, once candidate components are identified, they can be retrieved in the component library to confirm whether these components exist. If the component already exists, it can be retrieved from the component library for reuse. If a candidate component does not exist in the component library, then a new component must be developed. After the new component is successfully developed, it can be used to construct the target system on the one hand, and it can be stored in the component library on the other hand [17, 21]. In recent years, component-based software engineering methods have emerged, and software process models based on the component assembly have also emerged, and have gradually been widely used. The model has 6 stages to enhance the use of object-oriented technology, encapsulate both data and algorithms, and reuse components/classes by assembling the correct components.

6.7. The Prototype Model

The prototyping model is the most popular SDLC model. Prototyping is identified as the process of developing a working replication of software that has to be engineered. This process is an iterative process, which can avoid the invisible product during the development process of the waterfall model [17]. When the customers do not know the exact project requirements beforehand, this model is used. The feature is that a prototype of the end product is first developed, then the prototype is tested and refined as per customer feedback repeatedly till a final acceptable prototype is achieved for the final product. The prototype model is proposed in response to the difficulty of determining the software system requirements at the initial stage of software development. It draws on the architect's experience in designing and building prototypes, enabling software developers to quickly develop a prototype based on the initial needs of customers [22].

The prototype model uses the prototype realization technique and gradual refinement techniques. Using prototype realization techniques can quickly realize a practical system preliminary model, for developers and users to communicate and review, in order to more accurately obtain user needs. The gradual refinement technique is to gradually improve the prototype for its modification. It runs each time after user review and is repeatedly recognized by both parties.

7. Conclusion

The software life cycle model is more expressive, more in line with human thinking patterns, belongs to the human level of abstraction, and is a logical entity rather than a physical entity. The special nature of software products and the limitations of human intelligence lead to the inability to deal with "complex problems". The concept of "complex problems" is relative, and once people adopt advanced organizational forms, development methods, and tools to improve the efficiency and capability of software development, new, larger, and more complex problems are presented to them. Our study shows to promote the development of the software technology industry, and the technology gradually matures. During the operation and use of software, it does not have aging problems as hardware. But it has the problem of degradation and must be modified and maintained several times. We conclude it is especially important to choose a suitable software life cycle model to extend the life of software according to the specific conditions of different software. The selection of the software life cycle directly affects the project's lifetime. We hope that we can use its principles wisely in software development and take advantage of them.

Acknowledgements

I am very grateful to my advisors and professors for their encouragement and guidance. I would like to thank my family for their support of my studies. I would also like to thank my friends and classmates for giving me a warm friendship.

References

[1] Zhang Dian. Research on the development status and countermeasures of modernization technology of computer software engineering, Popular standardization, 2020 (16): 47-48.

[2] Wang Q. The application and development trend of computer software development technology, Science and technology innovation and application, 2021, 11 (28): 176-178.

[3] Bai Baoqi. The application of computer software development and its future development trend, Computer Knowledge and Technology, 2021, 17 (17): 53 54. DOI: 10.14004/j.cnki.ckt.2021.1552.

[4] Zhou C, Li GH, Chen H, Sun XJ. Interpreting the development status and countermeasures of modern technology of computer software engineering, Computer Knowledge and Technology, 2021, 17 (07): 242-243. DOI: 10.14004/j.cnki.ckt.2021.0804.

[5] Yang, Shuhui. The application and development situation of computer software development technology in the new era, Network security technology and application, 2020 (06): 68-69.

[6] Zhong Kun. The application and development trend of computer software development technology in the new era, Network security and information technology, 2022 (02): 28-30.

[7] Chen Q. Models and their applications in Web development, Information and Computer (Theory Edition), 2014 (12): 176.

[8] Wei Anruo. Analysis of the development trend and application of computer software development technology, Information Record Materials, 2021, 22 (11): 167-168. DOI: 10.16009/j.cnki.cn13-1295/tq.2021.11.078.

[9] Kong Xiao. Common software life cycle models in software engineering, Electronic Technology and Software Engineering, 2017 (14): 58.

[10] Wang Yumei. Software product life cycle planning and implementation process, Electronic Technology and Software Engineering, 2018 (21): 37.

[11] Li HX, Wang L, Li Z, Wang YB. Research on software life cycle quality evaluation methods, Computer Measurement and Control, 2022, 30 (08): 264-268+295. DOI: 10.16526/j.cnki.11-4762/tp.2022.08.041.

[12] Peter Morville, Louis Rosenfeld. Web Information Architecture: Designing Large-Scale Web Sites, Chen Jianxun, Translation. Beijing: Electronic Industry Press, 2008, 53-191.

[13] Wang Highest. Research on the quality metrics system of the whole life cycle of software projects in banking industry, Shandong University, 2014.

[14] Hu ZQ, Shang YJ. Analysis of software lifecycle reliability factors, Computers and Networks, 2017, 43 (10): 70-72+75.

[15] Research on Software Unit Testing and Test Case Design Methodology, Proceedings of the 17th Annual China Aviation Measurement and Control Technology Conference, 2020: 246-248. doi: 10.26914/c.cnkihy.2020.027598.

[16] Wang Wendong. Analysis of web software system testing application based on B/S architecture, Software Guide, 2016, 15 (08): 137-139.

[17] Yang Yang, Liu Quan. Software system analysis and architecture design, Nanjing: Southeast University Press, 2017.

[18] Zhou Su, et al. Modern software engineering, Beijing: Machinery Industry Press, 2016.

[19] Wang Xuemei, Zhang Chunhai. An improved software development model-combination model research, Software Guide, 2018, 17 (11): 52-55.

[20] Guo, Lian-Ming. Talking about waterfall model and its limitations, Science and Technology Outlook, 2016, 26 (06): 172.

[21] Sun Xiufang, Jiang Kai. Localization practice based on GJB5000A Level 2 software lifecycle model, Standards Science, 2020 (10): 72-76.

[22] Zhao Shujun. A life-cycle model for software development on bombs, Modern Defense Technology, 2020, 48 (06): 48-52+66.