Research on Software Life Cycle Model Suitable for Aerospace System Engineering

Chuyang Dong, Haihong Fang, Hongjie Zhang, Yuexi Wang and Xianqing Ling
Beijing Institute of Aerospace Long March Aircraft
Beijing, China
6661289@qq.com

2021 IEEE 12th International Conference on Software Engineering and Service Science (ICSESS) | 978-1-6654-1506-4/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICSESS52187.2021.9522355

Abstract—**China's aerospace products are known all over the world for their high efficiency and high quality, and aerospace software plays an important role in aerospace system engineering. However, the traditional Software Life Cycle Model is not suitable for the high-efficiency and high-quality development requirements of aerospace products. This paper summarizes the characteristics of the aerospace system engineering and the characteristics of aerospace software, and proposes a rapid iterative development model based on mature technical solutions. Using this model to develop aerospace software products can solve efficiency and quality issues at the same time, opening up new ideas for aerospace software development. And the effectiveness of the model has been proved through practice.(Abstract)**

Keywords-aerospace software; Software Life Cycle Model; rapid iterative development model (key words)

I. INTRODUCTION

A system is composed of a number of components that interact with and depend on each other, and it is an entirety having specific functions. Moreover, the system may be a component of a larger system to which it belongs [1]. The system mentioned herein refers to the software as a part or several parts, interacting with other parts or between them, forming a whole with certain functions. The development of aerospace products is the most typical system engineering. Therefore, the focus of this paper is to establish a software life cycle model to make software development and management more suitable for the requirements of aerospace system engineering.

The traditional concept is that a successful project is to achieve the project goal that meets the quality requirements within a limited time and limited resources, but this idea is too simple. For example, for a game software project, the project invested less manpower within the planned time and realized a game product with almost no defects. However, after the listing, it was found that the product positioning was vague, difficult to promote, and the market reacted indifferently. Such a project is not a successful project, and such a project management goal should not be the goal that project management should pursue. Therefore, time, cost, and quality are just three constraints. They are not the most important factors for judging the success of a project. Today, the International Project Management Association (IPMA) basically defines project management as: "Successful project management is the recognition and appreciation of project management results by project stakeholders."

Professor Qian Xuesen pointed out in 1978: "'System Engineering' is the scientific method of planning, researching, designing, manufacturing, testing and using the organization and management of 'systems', and it is a scientific method that has universal significance for all 'systems'[1]. System engineering emphasizes the coordination and mutual cooperation between the whole and the part, and emphasizes the optimal operation of the whole. The software discussed in this article is just an important part of the system, and its own function and relationship with the whole affect the function and performance of the entire system. Therefore, this article believes that the most important factor for the success of software project management in system engineering is "let the software play the most appropriate role in the system and make the system functions the most coordinated".

II. THE POSITION OF THE SOFTWARE IN THE SYSTEM

Generally, the position of software development in the system is simply represented as shown in Figure 1.

Figure 1. The position of software development in the system

Therefor Fig.1 only shows the source of the software requirements and the delivery object of the software, but does not specify the specific development relationship between the software and the system.

In fact, the design work of all links is carried out almost at the same time. The development of aerospace systems requires multiple iterations of design and verification to ensure that the solution is feasible or optimal to save test costs. Therefore, the relationship between software and system development should be shown in Fig. 2.

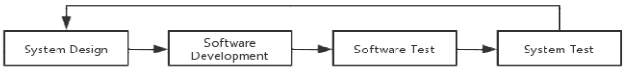

Figure 2. The relationship between software development and space system

Software and systems are more like a process of simultaneous development. The faster the software development responds to system requirements, the more timely the system verification, the faster the system maturity, and the lower the cost. It can be seen from this that the development of aerospace software is not only for delivery, but also requires continuous optimization and continuous iteration to optimize the development and verification of system functions. Software changes are not harmful, but are beneficial to the system and to

978-1-6654-1506-4/21/$31.00©2021 IEEE

the development of the software itself. We should embrace best to reduce the cost of changes. So how to develop software to better adapt to the role of software in system development? For this reason, this paper proposes a software life cycle model suitable for the development of aerospace systems.

III. Research on Software Development Model

There are many commonly used development models, such as waterfall models, prototypes, incremental models, spiral models, and iterative models[2]. Unfortunately, although these models are developed and developed to meet various development models and development needs, these models are not suitable for aerospace system engineering.

There are many characteristics of software development mode under system development that are different from ordinary software development, but the following three characteristics are particularly obvious: 1. The software scale is not large, most of the system requirements can be determined at the beginning and the horizontal tracking relationship is simple, but it needs to be gradually improved. A large number of improved designs come from the simulation verification results of the system itself, and partly from software operation feedback; 2. The system requirements are short-term and are the key link in model development. Once the system requirements are determined, it is hoped that the software products will mature as soon as possible to meet the requirements of high efficiency; Very high requirements on the quality of software products. Based on the above characteristics, briefly explain and analyze the reasons why the commonly used development models are not suitable for software development under system development.

A. Waterfall model

In 1970, Winston Royce proposed the famous "waterfall model". The waterfall model is an important step for people to get rid of the software crisis. The software development process is divided into requirements analysis, software design, software implementation, software testing, software maintenance and other stages, each stage must deliver qualified documents, forcing developers to adopt standardized methods to review documents, one stage only after being qualified can the next stage of development activities be carried out. The waterfall model is a document-driven model, which enables software development to be rule-based and to a certain extent guarantees the quality of software products.

However, the waterfall model is an ideal linear model. It requires a complete requirements from the beginning. It is afraid of feedback in user testing and fear of changes in requirements. Moreover, when the development cycle is tight or shortened, the activity that is often compressed is software testing, which makes it difficult to guarantee the quality of software and brings hidden dangers to the quality of the entire software product.

For areospece system, it is impossible to achieve the fullest requirements at the beginning of software development, and changes in requirements are also inevitable. The huge change cost organization under the waterfall model is hard to accept. changes, respond to changes as quickly as possible, and try our

B. Rapid prototyping model

The rapid prototyping model is essentially a do-it-yourself model. The focus is on rapid establishment, providing users with products that can be tried or used as soon as possible, and then quickly modifying the prototype according to demand changes, repeating this many times until the user needs are fully realized. This model overcomes the shortcomings of the waterfall model and effectively solves the problems of ambiguity and changing user needs.

However, a rapidly established system structure coupled with continuous modifications may result in poor product quality. Therefore, the rapid prototyping model is divided into discarded type and non-discarded type. The non-discarded type is as mentioned above, through repeated iterations, it eventually becomes a deliverable product, but there are inherent quality risks. Once the user needs are determined, the software must be redesigned. Discarded prototypes are only used for demand development, and the development of software products is also faced with the development model selection, and a large amount of work of developers is discarded, so the development cycle is lengthened.

Under the requirements of system development, it is required that once the system requirements are determined, the software products should mature as soon as possible and require high product quality. Therefore, neither discarded prototyping model nor a non-discarded prototyping model is applicable.

C. Incremental model

Incremental model provides a method for the grouped realization of software systems that have a clear system requirement priority. The latter version of the software system developed using the incremental model adds new functions on the basis of the previous version until all functions are realized. Usually, the first component completes the basic and core functions provided by the software.

However, the integration of each new component into the existing software structure will inevitably affect the originally developed product, so a good interface must be defined for the incremental component. Otherwise, the incremental model is easy to degenerate into a doing-and-modifying model, so that the control of the software process loses its integrity.

Under the requirements of system development, it is difficult to clarify all the requirements and priorities of the system initially, and secondly, the first component that realizes the core requirements may not have good scalability, and subsequent incremental development may bring hidden dangers to product quality. . Therefore, the application limitations of the incremental model are also obvious

D. Other models

There are other models, such as spiral model, W model, etc[3]. The spiral model is suitable for large-scale software development and emphasizes risk analysis. The W model is an improvement of the waterfall model, emphasizing verification and confirmation, but like the waterfall model, it is not easy to handle situations where demand changes frequently.

IV. ITERATIVE DEVELOPMENT MODEL BASED ON MATURE SOLUTIONS

In summary, in order to better solve the problem of mismatch between the development model and the system development goal, this paper proposes an iterative development model based on mature solutions. This model runs through the entire life cycle of aerospace system development, and can simultaneously solve the problem of adaptability of software development and system development and the maturity of software products.

However, the traditional aerospace software development model has its significant advantages, inherent characteristics and disadvantages. Advantages, rapid response to changes in demand, relatively short software delivery cycle from task to task, meeting high efficiency requirements, sufficient system verification, and certain assurance of software product quality; inherent characteristics, gradual improvement of requirements; disadvantages, lack of product planning, random design,hidden quality hazards, and poor product maintenance[5].

Therefore, based on the inherent characteristics, the advantages and the disadvantages, the iterative development model based on mature solutions is defined as shown in the figure below.

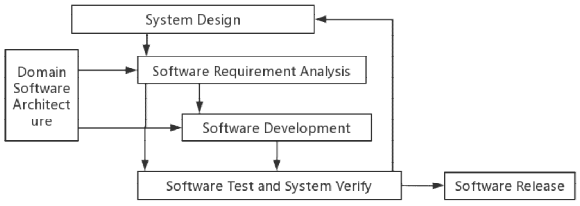

Figure 3. Iterative development model based on mature solutions

The model has the following characteristics.

- The domain software architecture is suitable for software development in this field. It is a mature solution for development in this field. Domain software architecture is the first component of software for incremental development[6]. The first component is not necessarily the most important function of the software, but the common requirements of domain products and the preset interfaces based on extensible requirements;
- In the process of software development/maintenance, if it involves the modification of the main software architecture, the modification plan must be determined through organizational discussion, or for special modification, or upgrade to the first component, so as to promote the continuous improvement of the solution of this type of product;
- The implementation and verification of demand additions, modifications, and deletions are basically consistent with the requirements of the iterative development model, but they are more manifested in rapid and non-fixed cycles, because software products are generally not on the critical path in the development of aerospace systems, but Its role may be reflected in many parts of the development process, so rapid and relatively random response to changes in demand is an important attribute supporting the development of system functions.

The advantages of this model are: based on a mature solution, in the process of quickly responding to changes in requirements, it avoids the quality risks caused by frequent changes to the software architecture. It is capable of responding to the rapid demand changes and ensuring the high quality of product development[4].

First, on the basis of project development experience, the organization has refined the public demand and variable demand of a certain type of product, has rich experience in the development of this type of product, specified the design specification of this type of product, and developed the first of this type of product. Component; Secondly, the scale of the software to be developed is not large, usually about 10,000 lines or less, so that the risk of the first component is less refactored; Finally, the horizontal tracking relationship between requirements is simple, and most of the functions can be verified in the system.

So, in the aerospace system, what proportion of the software that meets the above model selection requirements? As we all know, aerospace system engineering is a traditional technology industry. Embedded software is the main component and relatively has single function. Taking a certain type of rocket as an example, 91.3% of the software have are about or less than 10,000 lines, and basically all of them are traditional fields. There are good conditions for refining common requirements, and mature solutions can be formed.

V. APPLICATION OF ITERATIVE DEVELOPMENT MODEL BASED ON MATURE SOLUTIONS

The following will take a certain type of rocket positioning and return software as an example to introduce the development of the development work.

A. Establish a mature solution

The function of a certain type of rocket positioning backhaul software is to process the received inertial combination data and receiver data, and output the real-time position and attitude of the spacecraft. Generally, there are two main functions in this software, that is supporting system testing and supporting data processing. First of all, the software product is an embedded software, therefore parts of its software requirements comes from system requirements and parts from hardware interface requirements. Summarize the characteristics of software in this domain: common requirements include Kalman filter processing, initial alignment, navigation solution, data interface reliability processing; changeful requirements include system testing, timing processing; variable requirements include hardware support packages, data reception. Based on the above analysis, the software architecture design is shown in Fig.4. The positioning backhaul software is divided into driver layer, protocol layer and application layer. The driver layer mainly constitutes the hardware support package and data reception; the main function of the protocol layer is the reliability processing of the

data interface; and the Kalman filter processing, initial alignment, navigation solution, system testing, and timing processing are all classified as the application layer.

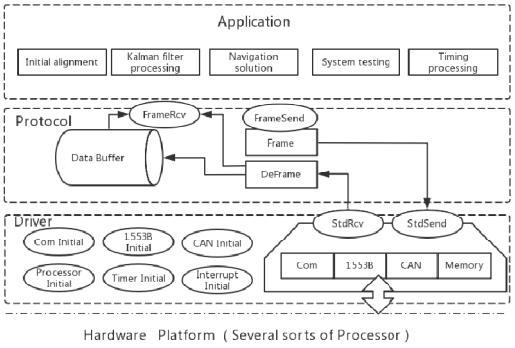

Figure 4. software architecture

This architecture and some reusable modules, like Kalman filter, Navigation solution, Processor Initial, Com Initial, 1553B Initial, CAN Initial, Timer Initialand so on, are composed of the first component of software, other software requirement are developed based on this first component.

B. Carry out iterative development of positioning backhaul software

The core principle of iterative development project management is rapid change and rapid participation in system verification. First, clarify the system's requirements for software functions. General system development is divided into system design, Electronic equipment production, equipment ajust, equipment test, semi-physical simulation and system verify. Software development activities include software development and testing activities. The testing activities are divided into drive testing, basic function testing, algorithm testing, and all software functional testing. As a result, the relationship between the positioning backhaul software product and the development of the system is shown in the Fig. 5 below.

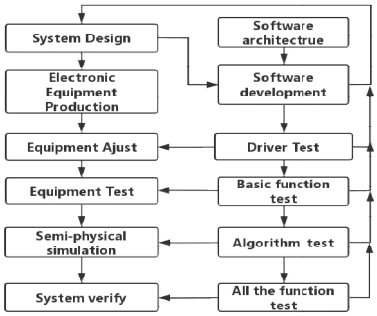

Figure 5. The relationship between the development of positioning backhaul software and system development activities

System designers put forward software design requirements suitable for the system according to the task background and project working conditions. The software design requirements roughly include hardware working environment, interfaces, algorithms, and test functions. Often the hardware working environment and external interface entities can be determined first, and the hardware is put into production and the design of the hardware support package is carried out at the same time. At this time, the algorithm personnel are still working on the algorithm design, and the data interface of the system can also be gradually determined at this stage. Before the electronic equipment production is completed, the software designer can complete the driver layer of the software development, and begin to carry out the algorithm and data interface implementation and testing. With the gradual improvement of requirements, the software is gradually improved and designed and tested. The software architecture ensures the right way to realize these requirements. Through the activities in Fig. 5, the software product provides the greatest support in terms of progress and quality to the system development.

C. Summary of research results

In the six-month development cycle of the positioning backhaul software, a total of 46 software requirements proposed by the system were received, including 25 original requirements, 21 additional requirements, and 14 response requirements. The test coverage of software requirements function, performance, and interface were 100%. After the system (including software) had been verified, the software version released the next day. The software requirement realization rate is 100%, and the number of quality problems after release is 0. The influence of software development on the system development cycle is almost negligible.

The use of a rapid iterative development model based on mature technology schemes truly realizes the matching of the software development process with the system development goals, and has significance for the development of space system software development activities.

VI. CONCLUSIONS

Aerospace software, as an important component of aerospace systems, is playing an increasingly important role. As the development cycle of aerospace systems is getting shorter and shorter, the proportion of software is increasing, but the product quality requirements are getting higher and higher. It is no longer possible to meet the requirements of high, fast, good, and economical only by being careful and careful. The software life cycle model proposed in this paper for aerospace system engineering can most quickly respond to system development needs, support rapid verification of system schemes, and ensure the quality of software.

REFERENCES

[1] Qian Xuesen ,"On System Engineering", 1982,pp.11.

[2] Wenhong Liu, “CMMI-based software engineering implementation: an advanced guide”, 2015, pp.68–83.

[3] Wang Lei, "A Value Driven Software Testing Model Based on Agile Development"[D], Beijing University of Posts and Telecommunications , 2014,pp.16.

[4] Li Jing, "Research on the Process Management of J Enterprise Software Project Based on the Scrum Model"[D], Yanshan University,2017,pp.21.

[5] Jiao Yuting, "Research on Software Cost Estimation Model"[D], Beijing University of Technology, 2013, pp.26.

[6] Jia Niyun, "Research on Model Traceability in Software Product line"[D], Hunan University, 2014,pp.6-12.