Inapplicability, Redundancy and Uncertainty of UML

Huachen Zhang

Information technology centre, Zunyi Vocational and Technical College, Zunyi City, Guizhou Province, 563123, China
17710180969@163.com

Abstract. Three problems i.e. inapplicability, redundancy, uncertainty of UML are discussed. In order to solve these problems, some ideas are given, including using business variables instead of classes, using system discrete events instead of use case diagrams, using requirement function instead of statechart diagram. New life cycle model is studied. New core ideas are proposed, including that requirement can not be refined step by step, and white box testing and black box testing are no longer differentiated but white box branches and black box branches are distinguished. Finally, the next step work is given.

1. Introduction

UML[10][11] is a unified modeling language or standard modeling language. It began as an OMG standard in 1997. It is a graphical language that supports modeling and software system development. It provides modeling and visualization support for all stages of software development, including from requirements analysis to design. The development of object-oriented analysis and design methods [7][8][9] has had a climax in the late 1980s and mid-1990s. UML is the product of this climax. It not only unified the representation of Booch, Rumbaugh and Jacobson, but also further developed it and eventually unified it into a standard modeling language accepted by the public. However, when UML is applied, there are still several problems that are easily overlooked and serious, such as inapplicability, redundancy, uncertainty, etc., which will greatly reduce the efficiency of software development.

2. A few problems for UML

2.1 Inapplicability for system requirement analysis

UML is based on an object-oriented approach that includes concepts such as objects, classes, messages, inheritance, encapsulation, polymorphism, and abstraction. Understanding and mastering these concepts is not an easy task. So strictly speaking, it is very inappropriate to use object-oriented methods to perform system function analysis, which complicates the problem. System-level user/requirements analysts should not see concepts such as "class", that is, users/requirements analysts should not see/care whether the system is implemented using a structure-oriented or object-oriented approach. In order to describe what the system is going to accomplish, it is absurd to require user/requirements analysts to learn and master many of the above concepts. In addition, if UML is used for system function analysis and software design is implemented in C language, then the mapping of the two is not very natural although it can be done. Beyond doubt, UML is only suitable for software design. So, there should be more direct, simple and clear methods for system requirements analysis.
2.2 Redundancy of diagrams
From the completeness point of view, UML mainly requires three types of graphs, namely use case diagrams, class diagrams/object diagrams, and statechart diagrams. Sequence diagrams, cooperation diagrams, and activity diagrams only capture parts of the above three graphs from different angles, making the business process visible from multiple perspectives. In other words, these three diagrams are redundant. The following three points are worth thinking about. First, adding these three diagrams will bring more workload; Second, it's easy to create inconsistencies between various types of diagrams. Third, the construction of the statechart diagrams is more in line with the idea of encoding.

2.3 Uncertainty of the degree of the diagrams description
The degree of description of the details of the statechart diagrams has not been clearly defined, so are the other diagrams. What content needs to be detailed? What content needs to be rough? Key issues have not been resolved. Many practical engineering examples spend a lot of time drawing UML diagrams, and these diagrams bring little value. If the description is too detailed, it is equivalent to doing what was done at the encoding stage. If the description is too rough, large rework may occur at the coding stage. Of course, not only does UML have this uncertainty problem, but so do structure-oriented methods [5][6].

3. Basic ideas for solving problems

3.1 Using business variables instead of classes
Compared with the concept of "classes", business variables are more direct and easy to understand. Business variables are the carriers of function implementation, and each function point is implemented by one or more business variables. For example, in order to complete the function of recording human information, perhaps at least the business variables of name, birth date, sex, height, weight, etc. need to be constructed.

3.2 Using system discrete events instead of use case diagrams
System discrete events mainly include the following categories:
- Click menu / button / dialog box.
- Click / drag mouse.
- Message communication.
- Input parameter.
- Timer overflow.
These discrete events are directly perceived by the system user/system requirements analyst and are not associated with the object-oriented approach. The handling of discrete events will replace the use case diagrams.

3.3 Using requirement function instead of statechart diagram
The requirement function eliminates the uncertainty of the statechart diagrams and describes the business variables as the core, with the following two characteristics:
- Accurately describes the rules of change for business variables.
- Does not describe how to implement these changes in business variables.

4. Life cycle model

4.1 Basic life cycle model framework diagram
The basic life cycle model framework diagram is shown below:
The basic life cycle model is divided into 9 phases:

- Phase I, i.e. description of original function requirement
- Phase II, i.e. system modeling
- Phase III, i.e. software modeling
- Phase IV, i.e. function modeling
- Phase V, i.e. encoding
- Phase VI, i.e. function function test
- Phase VII, i.e. softwareing function test
- Phase VIII, i.e. system function test of discrete
- Phase IX, i.e. integrated system testing

Phase I describes the original system function requirements and system test strategies, which are mainly derived from the market; Phase II performs the accurate system function requirement design/requirement analysis, and gives the discrete event system function test cases. These are done by business experts/formalization experts; Phase III performs the accurate software function requirement design/requirement analysis, and gives the software function test cases. These are done by software architecture experts; Phase IV performs the accurate function function requirement design/requirement analysis, and gives the function function test cases. These are also done by software architecture experts; Phase V completes the encoding works, which are divided into two parts i.e. internal project group functions and external outsourcing functions, and are done by software algorithm experts/formalization experts; Phase VI performs function function test; Phase VII performs software function test; Phase III performs system function test of discrete event; Phase IX performs integrated system testing, which includes combination function test, reverse thinking function test, etc..

Three types of requirements designs/requirements analysis have the same three characteristics: First, the key is to define the requirement function(involving one or more business variables); Second, only the white box branch can appear in the requirement function(the definition of the white box branch and the black box branch will be described in Section 5); Third, for visualization, the requirements function is represented by a tree layer.

When three types(discrete event system function, software function, function function) of test cases are constructed, they all follow the same three basic points: First, doesn't question whether the requirement design/requirement analysis are right, and only determine whether the implementation meets the requirement design/requirement analysis; Second, due to the large test space, it can not be proved by a limited number of test cases that implementation must meet the requirement
design/requirement analysis; Third, system test and function test no longer have the difference between white box testing and black box testing (This will be explained in detail in section 5).

4.2 Comparison with waterfall model
The new model, like the waterfall model [1], is V-shaped. First, each "requirement design/requirement analysis" phase corresponds to a "test" phase; Second, the encoding phase begins after all "requirement design/requirement analysis" phases are completed; Third, each "test" phase begins after the encoding phase is completed. The above three are combined to form a V shape.

The main differences between the new model and the waterfall model are the following. First, the new model emphasizes that the system function requirement design/requirement analysis, the software function requirement design/requirement analysis, and the function function function requirement design/requirement analysis must all be carried out in a "accurate" manner. The waterfall model does not emphasize "accuracy". It is also because the former emphasizes accurate requirement design/requirement analysis, so it is possible to give accurate test cases at each stage, and these test cases do not and should not change after the encoding is completed. In contrast, the vast majority of test cases at all stages are discarded after the encoding is over in the waterfall model. Second, the new model emphasizes the function testing of discrete event systems, which is not available in the waterfall model. Third, the new model divides the function implementation into project group function implementation and outsourcing function implementation. This division is not in the waterfall development model. It is essentially due to the inevitable result of accurate design/analysis of function functions.

4.3 Comparison with agile model
The life cycle model supporting iteration is shown in figure 2 and figure 3.

![Figure 2. Iterative life cycle model framework diagram I.](image-url)
The new model, like the agile [2] [3] [4] development model, highly agrees with the concepts of requirement division, requirement classification, iterative development, and continuous integration. Of course, the division of system function requirements in the new model and the "Story" requirement division in the agile model have both the same points and different points. The same point is all from the user's point of view, emphasizing priority. The difference is that the former emphasizes the outputs and change rules of business logical variables (see in section 5) brought about by discrete events, while the latter emphasizes the value of identifying requirements to users. Obviously, the latter concept is more vague and more difficult to grasp. Also, the iterative development in the new model and the iterative development in the agile model have both the same points and different points. For those developments that have unclear needs and are constantly interacting with external customers, the former, like the latter, will use iterative development methods to construct the minimum system with the customer to identify the requirements implemented. Then through the version of the iterative way to continuously modify, improve, enrich the requirements of functional points. The first iteration is shown in Figure 2. For communication protocol software development, because the requirements are relatively clear and the links between the requirements are relatively large, if the former adopts iterative development, it will be different from the latter, that is, all system function requirements design/requirements analysis will be completed at once. This would be more efficient overall. The second iteration is shown in Figure 3. In particular, the new model naturally supports continuous integration because it supports outsourcing function implementation.

There are a few differences between the new model and the agile model. First, the former emphasizes three levels of accurate requirement design/requirement analysis, but the latter emphasizes only sitting together. However, simply sitting around does not necessarily make a real sense of the requirements, because verbal expression is difficult to describe complex logical processing branches. Second, the former divides the function implementation into the project team function implementation and the outsourcing function implementation. This division is also not in the latter, which is completely incompatible with the latter's promotion of "team members should sit together." Third, as mentioned above, the latter emphasizes that developers should sit together and discuss together to determine system function requirements. However, the former advocates the classification of developers and encourages different experts to take charge of different things. The former divides experts into three categories, namely business experts, software experts (software architecture experts, software algorithm experts), mathematics experts/formalization experts. The business expert is an expert in the business field and is responsible for system function requirements, for the complex
processing branch of the business, and for system test cases. Business experts can not understand
hardware and software; Software architecture experts are familiar with both business knowledge and
software data structure, and are responsible for programming language selection, platform selection,
data structure selection, function function division; Software algorithm experts are proficient in data
structure and algorithms, and mathematical experts/formal experts are proficient in formal proof
methods. Fourth, the latter emphasizes TDD, but due to the large test space, the test can only be false.
Therefore, the former encourages the introduction of formal methods in the system function analysis
stage and outsourcing function implementation stage, which can further improve the quality of
software development(see in section 5).

5. Subversive ideas

5.1 Requirement function

5.1.1 Requirement can not be refined step by step. Unlike traditional idea, the new idea holds that
requirement can not be refined step by step.
requirement, software function requirement and function function requirement, these three requirements are described in the same way.

5.1.2 Business logic variables and business physical variables. Business logical variable is the carrier of system function realization. Each system function point is realized by one or more business logical variables. For example, a book management system that needs to implement the "borrowing of books" function requires at least five business logical variables to record the key contents of each book, such as the number, name, author, publisher, and status. As shown in Figure 5, the content of the business logical variable is divided into two parts, the first part is the control content/state content, and the second part is the information content. Both of these contents may be dynamic. Take the "borrowing of books" function of the book management system as an example. The contents of the business logical variables of the book number, name, author, and publisher belong to information content. As far as this part is concerned, once it is determined, it will not change again. So what's the control content? It's the state of the book. For example, the book has at least the following states: ordering, cataloguing, collection, lending, repayment, etc.. It needs to be emphasized that business logical variables do not involve any specific implementation. For example, variables are implemented by character type or integer type.

![Control content information content](image1)

Figure 5. Business logical variable composition.

The business physics variable is the specific implementation of the business logical variable. As shown in the figure 6, the content of the business physics variable is divided into three parts. The first part is the control content associated with the business physics variable, and the second part is the control content/state content. The third part is the information content. All three parts may be dynamic. For example, a server can establish TCP connections with thousands or more clients, but the server does not know in advance how many clients will initiate TCP connections. So how does the server organize these TCP connections? The first part of the content reflects the association between "these TCP connections", that is, through this part of the content one TCP connection can be indexed to another TCP connection. In addition, there is the possibility of cross-indexing between different types of business physics variables.

![Associated Content Control content information content](image2)

Figure 6. Business physical variable composition.

If software development is implemented by C language, the business physical variables correspond to the structure or union; if software development is implemented by Java language, the business physical variables correspond to the class; whether the structure or the class, they correspond to the same business logic variable. That is, the same business logic variable can be mapped to multiple business physical variables according to different implementations.

5.1.3 System function requirement function. System function requirement function is triggered by system discrete events (see 3.2). System function requirement function is defined as:

- The requirement function is the integration of requirement design and requirement analysis. "How to do" corresponds to requirements design, and "what to do" corresponds to requirements analysis.
The "how" judgment branch comes from the combination of discrete event control content and business logical variable control content. The order of execution of the judgment branches reflects the "careful design" of the business experts. These judgment branches do not have an absolute execution order, but the good execution order and output correspond to a good user interface and user experience.

"What to do" describes the changes in the content of business logical variables and output under each of the above judgment branches, including the generation and deletion of business logical variables. Note that the content of the logical variables here includes both control content and information content. Output includes: pop up display box, pop up dialogue box, pop up input box, send messages to the outside of the system, etc..

The essence of accurate system function requirement design/requirement analysis is to describe the system function requirement function.

Discrete event system function test has the following characteristics:
- From a systemic point of view, no specific implementation of the procedure is involved.
- Take a single discrete event as the basic entry point for system function test.
- Verify that the changes of the related business logical variables and outputs meet the system function requirements design/requirement analysis.

In classical software engineering papers and books, when system function is tested, they generally only focused on the input and output of the system, and rarely (or even not) emphasized the change of business logical variables. Of course, this is also due to the fact that the change rule of business logical variables is not accurately given during the system function requirement analysis stage. So, how to determine whether the changes in business logical variables are in line with the system function requirement design / requirement analysis? It can be judged in three ways: The first is to judge by query (if there is a direct query operation); The second is to judge directly through the user interface perception; The third kind is judged by the discrete event combination and then using the user interface perception.

5.1.4 Software function requirement function. The starting point of the description of Software function requirement function is from the entry function. Software function requirement function is defined as:
- The requirement function is the integration of requirement design and requirement analysis. "How to do" corresponds to requirements design, and "what to do" corresponds to requirements analysis.
- The "How to do" judgment branch comes from the combination of input parameter (entry function) control content and business physical variable control content.
- "What to do" is to describe the changes in the content of business physical variables and output under each of the above judgment branches, including the generation and deletion of business physical variables.

There are three obvious differences between the requirement function definition of software function and the requirement function definition of system function as follows:
- The source of the "how to do" judgment branch. The former comes from the combination of input parameter control content of the entry function and business physical variable control content, while the latter comes from the combination of discrete event corresponding control content and business logical variable control content. But there is a correlation between the two.
- The process of creating and deleting variables. The former corresponds to the allocation and release of memory, while the latter is only a virtual existence and disappearance.
- Changes in variable content. The former needs to describe the "associated content" used for the index, while the latter does not need to be described because it is virtual.

Software function test has the following features:
• Entry function is as the starting point for software function test. In other words, when software function is tested, the entry function is called.
• Verify that the changes of the relevant business physical variables and outputs are in line with the software function requirement design/requirement analysis.
• Construct the software function test cases before decomposing the entry function function.

The essence of accurate software/function function requirement design/requirement analysis is to describe the software function requirements function.

5.1.5 Function function requirement function. There is no essential difference between software function requirement function and function function requirement function. It is only that the scale is different.

5.2 White box branch and black box branch
Traditionally, black box testing is also called functional testing or data-driven testing. Black box testing treats the program as a black box that can not be opened, regardless of the internal structure and internal characteristics of the program; White box testing is also called structural testing, transparent box testing, logically-driven testing, or code-based testing. White box testing has a complete understanding of the internal logic structure of the program and tests all logical paths. In the new approach, it can not be simply considered that "system function test is a black box test" and "unit test/function function test is a white box test", but system function test, software function test, and function function test all include white box part(blanch) and black box part(blanch). It is necessary to test whether the black box implementation violates the "what to do" in the requirement analysis at each branch of the white box, as shown in Figure 7, there are 6 main paths, as follows:

- Path 1: 2--->5 A
- Path 2: 2--->6 B
- Path 3: 1--->3--->7 C
- Path 4: 1--->3--->8 D
- Path 5: 1--->4--->9 E
- Path 6: 1--->4--->10 F

For example, in the case of path 3, i.e. 1 - > 3 - > 7, execute C, assuming that C completes the function of "making a business variable change according to a particular law". The 1--->3--->7 case is visible, but how to implement C is invisible.

Figure 7. White box part(blanch) and Black box part(blanch).
The white box branch has the following characteristics: First, the appearance of the white box branch will be displayed; Second, the white box branch appears in the requirement design/requirement analysis description of system functions, software functions, and function functions; Third, if you want to change the white box branch, no matter how it changes, the change will not be too large, and this change is more of a change in the order of execution.

The black box branch has the following characteristics: First, in the requirements design/requirement analysis description, the black box branch will not appear. Black box branch only appears in the encoding; Second, when the specific code is implemented, the black box branch generally appears with the loop statement; Third, different implementations, the difference between black box branches may be relatively large.

Orthogonal experimental design test method is a common black-box test method, which covers two sets of combinations of multiple variables with a minimum of test cases. Obviously, because of the white box branch, the new system test method will use fewer test cases than the orthogonal experimental design test method.

6. Next step work
The above is more theoretical exploration. On the one hand, it is necessary to verify the above ideas with rich practical software projects, and add and modify some details. On the other hand, a UML-like tool needs to be developed to better support the above theory.

References
[1] Royce. Winston. (1970) Managing the Development of Large Software Systems, Proceedings of IEEE WESCON 26 (August): 1-9.
[2] Mike Cohn. (2009) Succeeding with Agile: Software Development Using Scrum, Addison-Wesley Professional.
[3] Robert C. Martin. (2002) Agile Software Development, Principles, Patterns, and Practices, Pearson Education.
[4] Kent Beck. (2002) Test Driven Development: By Example, Addison-Wesley Professional.
[5] Edsger W. Dijkstra (1968), Go To Statement Considered Harmful, Communications of the ACM, Vol. 11, No. 3, pp. 147-148.
[6] Yourdon, Edward; Constantine, Larry L. (1979) Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design. Yourdon Press.
[7] Grady Booch. (1993) Object-Oriented Analysis and Design with Applications (2nd Edition), Addison-Wesley Professional.
[8] P. Coad and E. Yourdon. (1991) Object-Oriented Analysis (2nd Edition), Prentice-Hall.
[9] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, William Lorensen. (1994) Object-Oriented Modeling and Design, Prentice-Hall.
[10] Ivar Jacobson, Grady Booch, James Rumbaugh. (1999) The Unified Software Development Process, Addison-Wesley Professional.
[11] Mark Priestley. (2003), Practical Object-Oriented Design with UML (2nd Edition), McGraw Hill Higher Education.
[12] CLARKE E M, WING J M. (1996) Formal Methods: State of the Art and Future Directions[J]. ACM Computing Surveys.
[13] NASA JPL. Formal Methods Specification and Verification Guidebook for Software and Computer Systems. Pasadena. CA. USA.
  - VOL(1): Planning and Technology Insertion.[NASA-GB-002-95]. 1995
  - VOL(2): A Practitioners Companion [NASA-GB-002-97]. 1997
[14] John J. Marciniak. (2002) Encyclopedia of Software Engineering. Wiley,John&Sons, Incorporated.