應用測試於軟體發展生命週期

Testing In The Software Development Life Cycle

蔡博元  莊立文
真理大學  資訊管理系

摘要

在本篇研究中,吾等提出一個軟體系統發展生命週期模式,稱之為測試設計階段處理模式。此模式強調反覆的測試各設計階段,在軟體發展生命週期中的每個階段均安排作適當的測試。測試活動起始於系統的開發,並跨越所有的後續階段。一些系統發展生命週期模式指出測試工作乃在程式編碼階段之後才被執行,這樣的情形可能使得所發展的軟體,因未被充份地測試即匆促上市。在整個系統發展期間,每當某個階段工作完成,對應於該階段的測試工作亦應同時作業完畢。

關鍵詞:系統發展生命週期、測試、設計、反覆的測試設計

Abstract

In this paper, we present a software system development life cycle model, called the Test Design Stages Processed model (TDSP model), in which we emphasise that iterative test design stages should be processed during each phase of the software development life cycle. The activity of testing begins during system development and spans all subsequent phases. Some system development life cycles describe testing which is performed after the coding phase, but this may cause the software to be delivered without sufficiently testing. During system development, that a phase has been completed, the testing for the phase should also be completed at the time.

Keywords: Software Development Life Cycle, Test, Design, Iterative test design stages
1. INTRODUCTION

When testing is performed after coding has been completed, the time available for testing may be limited by the target date for project completion, especially if the project process has been poorly controlled. This may cause the software to be released without having been sufficiently tested. To certify the quality of the software, verification and validation tasks should be exercised in every phase of the whole process. Therefore, testing, which is a technique used to validate processes, must be distributed and performed in every phase of the software development life cycle. This means system development and test design should be carried out concurrently. Test design must be included at each phase of the system development, and measurement should occur at the end of each phase.

A software system development life cycle model, called the Test Design Stages Processed model (TDSP model), is presented in this paper. The model emphasises that iterative test design stages should be included in each of the phases of the software development. Thus, when a phase has been completed, the testing activities for the phase should also be completed at that time. Practically, the system design and test design processes in the system development projects are performed concurrently. To match the practice, the authors have proposed this model in software engineering teaching and training courses. This model can also be used to describe object-oriented system development. The emphasis here is that design, implementation and testing are interleaved activities which should be performed in an incremental fashion.

Four different software development life cycles are briefly introduced in section two. A testing policy, explained in section three, from which the TDSP software development life cycle is described. The TDSP life cycle is discussed in detail in section 4. Finally conclusions are presented in section five.

2. SOFTWARE DEVELOPMENT LIFE CYCLE MODELS

2.1 Waterfall Model

The waterfall development model (see Figure 1), which is attributed to Royce and is well known through Boehm [2,11], starts at the analysis phase and progresses to the design phase, through to the coding and finally the testing phase. In this model, it is necessary to revisit earlier phases of development. This is because the implications of a decision in a particular phase could not have been foreseen until it was worked through in later phases.

![The waterfall model](image-url)
2.2 Traditional V Model

The V model (shown in Figure 2) was developed using the waterfall model as its the basis [6]. In this model, testing is carried out in reverse order to software construction. However, both this model and the waterfall model have the problem that the testing tasks are not exercised until the program code has been generated.

![Figure 2 Traditional V model of software development](image)

2.3 Test Plans In V Model

The test plans in V (TPV) model in Figure 3 is similar to the V model in Figure 2, but the former shows the role of the test plan. The test plan is carried out at the system requirements phase and should be developed in detail as the software is designed [12]. The relationship between the test plans and the software process phases is that the test plan phases link the design phases to the testing phases. The test plan processes only produce the testing specifications. Testing is undertaken at each testing phase in the right side of the model. TPV model does not describe the iterative and incremental procedures.
2.4 Spiral Life Cycle

The spiral life cycle (see Figure 4), proposed by Boehm [3], follows neither bottom-up nor top-down development procedures. Analysis and design follow an incremental and iterative process. Each phase can provide information about the modification of the results at other phases including some previously completed phases. Verification and validation are performed at each design phase, but testing begins when coding is finished. The problem with this model is similar to that of the V model, the testing tasks are performed during the final phase of the life cycle.
3. TESTING POLICY

The design and testing of an application under development appears to be a difficult task, since no matter how good the methodology used to test an application during development, the application may be executed again with further test data to prove that it is fault-free when it has been completed. During development, however, testers can perform test work such as designing test methods, test drivers, test stubs and generating test cases, and so on. These can be used to test the application once it has been developed. Therefore, the end of the design process is the start of test execution, but not that of the test work. In other words, when one phase of a design has been completed, the testing for the phase should have been completed [14,15].

Based on this testing policy, test design work is performed following the specification rather than the program code. The former is also used as a reference document to develop the application. Therefore, a well-defined specification is more important for designing and testing applications. In test design, testers may follow the specification to create a simulation of the required application. The simulation of an application can provide an environment that can be used to certify the test cases. These test cases can also be used to test the real application once it has been developed.

Test execution and analysis attempt to evaluate whether the application achieves the level of quality required by executing the test cases and reporting the test results. Faults occurring in a developed application may indicate that the application is incorrect or that the test cases are inadequate. For example, after executing the test cases of the developed application and collecting the test results, testers may compare the test results with the expected results. The results of this comparison may be:

1. Errors unfound;
2. Errors found (error-exist). There could be:
   - incorrect application: test results are different from correct expected results,
   - incorrect oracle/expected result: the oracle is unable to analyse the execution results, and
   - incorrect test cases: test execution does not return a result.

If faults exist in the application, the designers should review the system design work or the testers should review test design work. Based on this testing policy, the proposed TDSP model shown in Figure 5 illustrates the testing design stages in the software development life cycle model.
4. THE TDSP LIFE CYCLE

The test design stage is separated from the system design and testing phases to form the TDSP model, see Figure 5. This consists of three stages and can be represented by:

\[
\text{TDSP life cycle} = \text{system design} + \text{test design} + \text{test execution}.
\]

The iterative design and testing processes in TDSP can also be represented by the following recursive expression, in which “|” symbolizes “or”:

\[
\text{process} = \text{system design} | \text{test design} | \text{test execution} | \text{null}
\]

\[
\text{process} = \text{system design} + \text{test design} + \text{test execution} | \text{process}.
\]

For example, if a unit design and test process is executed and test case errors are found, the tester needs to reconsider the test design and then perform test execution again. Therefore, the process comprises the detailed design, unit test design, unit test execution and the iterative unit test design, and unit test execution stages.

System design and system testing are performed incrementally in TDSP. The related modules that pass unit testing can be combined into an integrated subsystem. While the integration testing is being executed, the integrated subsystem may be waiting to join with
other related module(s) to enlarge the integrated subsystem. This incremental process is adopted in the TDSP model.

If any faults caused by the recently added module(s) are found during integration testing, this integration testing provides the designers with information in order to review the detail design process and to modify the incorrect/unsuitable module(s).

The TDSP software development life cycle is vertically divided into the three stages: system design, test design, and test execution. In addition, the model is horizontally divided into three phases:

1. requirements specification, system test design and system testing,
2. architectural design, integration test design and integration testing, and
3. detail design, unit test design and unit testing.

The output specification of each system design process is the reference document used by system designers to perform the system design process of the next phase, and by testers to perform the test design process of the current phase. For example, while testers are working in the unit test design process, the programmers simultaneously code the module (unit) programs using the module specification. When a module has been coded and its test design completed, the module can then be unit tested.

In this paper, the authors will focus on the test design stages of the three phases and the discussion of them will consist of verifying specification, test design, and test execution three parts.

1) Verifying specification
The TDSP model emphasises the specification that is the only output of each design process. There are no further processes before the specification having been verified to meet the requirement. The verified specification becomes a reference for designers and testers to do further system design and test design processes.

2) Test design
In the test design process, testers will involve: designing testing criteria and testing methods, generating test cases/data, expected results, and so on [10]. Of course, these work, which will be implemented to detect real programs, can be produced by following either specifications or programs. In order to reduce the system development time, test design and systems design should be exercised at the same time. Based on this, the test design of the TDSP model is called specification-based.

3) Test execution
One of the differences from other models is that only the test execution and test result analysis are employed at each testing process in the TDSP model. This is because the test work has already been finished at the test design stage. After processing the simulation testing, testers might be waiting for the related design products, such as modules, integrated parts or system, which are still developing, because test design work has been finished early. When the system design work is done, testers can reuse the test cases/data and expected results, which have been used in simulation testing, to test the real products. After test execution, if the products are fault-free, then this phase is completed. Otherwise, designers and testers should go back to review the designed work and the generated test cases/data and expected results.

4.1 Requirements Specification And System Test Design Phase.
The purpose of the requirements phase is to ensure that the users’ requirements are properly understood, and translated into design. Additionally, system testing is not simply the process of function testing the complete integrated system. System testing is exercised when the system has been installed on the target platform and is used to demonstrate whether or not
the system does meet its original requirements and objectives [8]. This testing task is an end-
user view to the system.

The goal of verifying requirement specification is to look for ambiguities, completion,
consistency, and reasonability, in addition, whether it is achievable, traceable and measurable
from a testing standpoint [8]. If the specification satisfies the user requirements, testers may
follow the specification to build up a simulation environment which may be called prototype.
The prototype is a solution at an early stage in order to prepare for later development. The
simulation testing in this phase is to validate the prototype, which is established according to
the specification, with the designed test cases/data in order to certify these test cases/data, and
expected results, because they will be used to test the real system.

Following the specification, testers can perform the test work at the early stage in each
phase. Therefore, the test design work and system design work can be exercised concurrently.
If the prototype can be carefully designed and actually tested, it may completely imitate
the final system. The acceptable prototype proves that the test work can be used to test real
system.

The purpose of system testing process is to check whether the system does meet its
original requirements and objectives. The test cases and data, which have been designed
earlier, can be reused to test the real system, or the prototype can be used as an oracle to
compare with the real system. If faults are found, then testers go backward to the
requirements specification process, see the outermost loop in Figure 5. When the real system
conflicts the correct and accepted prototype, this means faults exist in the real system.
Nevertheless, It is better to review the specification and test design completely again. If the
faults are caused by incorrect test cases, test data or expected results, then testers need to go
back to system test design process to modify test work. Contrary, if the faults exist in the real
system under testing, then system designers need to review all design work on a process by a
process level.

4.2 Architectural Design And Integration Test Design Phase.

Architectural design is the process of translating user requirement into the set of external
interface. Designers, in this phase, should identify subsystems for the whole system, establish
a framework for subsystem control and define subsystem interfaces [12].

The purpose of integration testing is to inspect the correct interaction of the parts (sub-
systems) in the software system under test and also to verify that the parts (sub-systems) are
working together correctly. In this testing process, testers may combine and test multiple units
gradually. The size of the tested parts (sub-system) is enlarged as the other parts have been
integrated for testing step by step. Integration testing is finished once the extended parts are
as large as the whole system. This testing normally occurs on the development machine.

Testers have built up interaction diagrams which explicitly show the interconnection of
several subsystems in top-down fashion or bottom-up fashion when process the simulation
testing. For every interconnection, testers may follow data-flow method, transaction-flow
method, or method-message path method and so on to design test cases, test data and
expected results, in order to test integrated parts [1,7]. During executing the integration
testing on the real applications (integrated units), testers may follow the approach which has
been used in the simulation testing, and gradually replace the stubs and drivers by the
modules which have passed unit testing.

Integrated parts, which have passed integration testing, may be waiting for other related
and tested module(s) joining into in this process. Once new module(s) has (have) added in,
the integration testing for this larger parts can be exercised. If there are fault-free results, then
this integrated parts are suspending and waiting for next new related and tested module(s). On
the other hand, if faults have been detected, then these faults may be classified into two:
The incorrect integration design, or incorrect integration test cases/data.

The new joined module(s) is (are) invalidated.

In this architectural design, tested modules and integration testing phase, there are two process cycles which are called small and middle process cycle, see Figure 5. The middle cycle process is when the detected errors are the sort of interface problems or integration test design problems, and the module codes should not be revised. The small cycle process is when the detected errors are not belong to the integration errors, but the tested modules (they may be the new added or may have existed in the enlarged parts already, but can not interact with the new added modules). Then these modules should be revised, because some modules’ errors which have not been found in unit tests, but may been uncovered during integration testing.

4.3 Detail Design And Unit Test Design Phase.

Detail design is the process of translating the design specification into a detailed set of data structure, data flow, and algorithms. The output of this process is the module specification that shows how the program is to be built [8].

The unit (module) testing focuses on the testing small building blocks of a program such as a class in OOP. The purpose of this testing is to discover contradictions between the unit specification and its real behaviour. Testers, in unit test design process, refer the module specification to design test work which can uncover faults within the boundary of the module, and can verify the correct operation of the methods in each module (unit).

Some of system development projects assign the responsibility of module (unit) testing to the programmers who also develop the module. Programmers, like ordinary people, do no like to admit to making mistake. Moreover, one of the purposes of testing programs is looking for misinterpretations of the specifications, therefore, the unit testing performed by testers is necessary. Because of programmers, who may misunderstand specification, may also test the programs with the same misinterpretations. Therefore, the “fault-free” programs may still have faults. The authors believe that qualitative testing should be done only by an independent group, testers, and not the programmers themselves.

Traditionally a unit test consists of structural (white-box) testing and functional (black-box) testing, in which there are statement, branch, and condition coverage [9]. In object-oriented systems, testers may perform testing based on the encapsulated state of the class object, this is called state-based testing [13,14]. In unit testing, there are several things need to test, such as module interface, data structure, boundary conditions, all independent paths, and error-handling paths, and so on. A module might be not a stand-alone program, so that driver and/or stub software must be developed for each unit test. A driver, similar a main program”, simply accepts test cases/data and then passes such data to the module under test, finally, prints the relevant results. Stubs replace modules that are subordinate (called by) the module to be tested.

During unit test execution, the test results will be generated. The test report will be generated by comparing test results with expected results (or oracle). If a problem occurs in the test execution, it may be caused by either incorrect program or incorrect test work.

Therefore, programmers and testers need to go back to detail design process to review the specification. If an incorrect test design occurs, testers need to modify the test work, otherwise programmers need to re-code the program.

In detail design, coding and unit testing iteration phase, each module is designed, coded and tested separately. When related modules have been coded and tested, then testers can go up to upper phase to integrate them as a sub-system and carry out the integration testing. At the mean time, some of modules are still performing in this phase, such as coding or testing.
5. SUMMARY AND FUTURE WORK

In this paper the authors presented an approach to separate test design work from design and test phases to propose a new software development life cycle, called the TDSP model. This can be divided into three vertical stages and three horizontal phases.

Before explaining the TDSP model, four different well known software development life cycles were introduced. The authors explained that unlike these methods the test work and the system design of the TDSP model are performed in parallel, therefore the test can be executed once the applications have been completed. To achieve this the verified specification is more important for designers and testers.

Test design and test execution stages in TDSP life cycle also show that test work is implemented iteratively and incrementally. Object-oriented software could be developed under any software development process model [5]. In testing, Booch[4] has suggested that the use of an object-oriented paradigm does not change any basic testing practices, such that unit testing, integration testing and system testing all have useful roles in object-oriented software testing. How to use the TDSP model to describe Object-Oriented software system development that emphasises an iterative and incremental development process is currently being addressed.

6. REFERENCES