Possibility of Reuse in Software Testing

"6th annual International Software Testing Conference in India 2006"

Manjari Gupta
Dept. of Comp. Engg., Institute of Technology,
Banaras Hindu University, Varanasi-221 005
manjari_gupta@rediffmail.com
Meeta Prakash
meeta_prakash@rediffmail.com

ABSTRACT

Testing of software is considered to be the most important component of software quality assurance process. The inherent errors that are left behind in a software code, will eventually lead to a lot of software failures. There is a possibility of applying reuse techniques in testing, as reuse is well known as a means for improving software quality and productivity. Unfortunately, software reuse and testing, both, are areas that are generally neglected by the professionals, during the development, because of various constraints and limitations. Reuse is useful and Testing is an important task, then it is obvious to get interested in and be concerned with the application of reuse in software testing. To get the maximum benefit of reuse, it should ideally be implemented in all the phases of software development rather than only at coding. Since testing phase is considered as the most difficult and time consuming, it may be possible to reduce the total software development cost, by applying reuse in the testing phase. This work identifies various possibilities of reuse in the testing process. Different kinds of reusable testing artifacts are highlighted, and the categorization of these artifacts in reuse repository is explained.

KEY WORDS Software reuse, Software testing, Test design patterns, Test environments, Test reuse repository.

1. INTRODUCTION

Quality assurance is an important issue in software development that is mainly achieved by testing among others like design reviews, formal specifications, model checking and inspection etc. In recent years, there have been significant strides in technologies supporting reuse-based software development, including the widespread adoption of design patterns, frameworks and component implementation technologies such as .NET and J2EE. However, reuse has not been much applied in testing phase. Testing process includes taking test requirements as input and preparing data for test execution, performing test execution, and evaluating test results. Testing normally requires 50% to 60% effort of total software development. Since testing is a difficult and time consuming activity of the software development there is a need to use, with other things, the integrated and automated testing tools. Testing involves exercising the program using real data to be processed by the program. The existence of defects is detected from the unexpected system output. Reuse has been considered as a technology that improves productivity and quality. Initially reuse was considered and used only at coding phase. But later it was found that only reuse at coding phase cannot solve the software crisis. And now reuse is being applied at each phase of software development. However, it is not being practiced very well. There are both technical and non-technical problems that inhibit organizations to practice it [1]. Since testing is hardest and time consuming activity in software development, possibility of reuse must be seen and applied at this phase. The idea is, to document and save the experiences that a tester gains during testing so that novices can take benefit from the experience of the work done already by others.

The idea of reuse has been very promising in terms of effort reduction during the software development phase and the subsequent software maintenance phase. It will be very useful for the software industry to reduce cost and enhance quality of testing by applying and practicing reuse. Patterns and frameworks for testing can effectively be used to achieve this aim. This paper is an attempt to demonstrate this philosophy of approach. The activities of test case generation, test case execution environment/tools, test deliverables generation, analysis of test deliverables all have a good deal of possibilities of reuse. The job of debugging, that is removal of faults, itself goes for reuse of test cases so as to be confident that the software works as per specification after corrections have been applied to it.

In the next two sub-sections, we describe software testing and reuse in brief. In section 2, we explain how reuse can be applied in different phases of testing process. Reusable test artifacts and their categorization are explained in section 3. The concluding section 4 summaries the idea and its usefulness.

1.1 SOFTWARE TESTING

In this section, we briefly describe all the testing methods. There exist a vast number of testing techniques and methodologies that help to achieve cost effectiveness for the testing phase. Two basic approaches to testing are: functional and structural [2]. In functional testing the structure of the program is not considered. The test cases are decided solely on the basis of the requirements or specification of the program or module, while in structural approach test cases are generated based on the actual code of the program or modules to be tested. Both testing approaches are complementary to each other. That is both must be carried out to test a system satisfactorily. The intent of structural testing is not to exercise all the different input or output conditions but to exercise the different programming structures and data structures used in the program. Control flow-based criteria, data flow-based criteria, mutation testing etc. are examples of structural testing approach. There are a lot of levels at which testing is considered by various researchers. A usual testing process goes through the following levels:

1. Unit testing
2. Integration testing
3. System testing
4. Acceptance testing.

When applying the well-understood levels of testing during the usage of test methodology, it becomes a difficult task.

Unit testing is called the first level of testing. It is essentially for verification of the code produced during the coding phase, and hence the goal is to test the internal logic of the modules. Consequently, mostly structural testing is performed at this level. The next level of testing is integration testing. The goal here is to see if the modules can be integrated properly, that is, many unit-tested modules are combined into subsystems, which are then tested here. The next level is system testing and acceptance testing. Here the entire software system is tested against requirements document. Acceptance testing is sometimes performed with realistic data of client to demonstrate that software is working satisfactorily. The internal logic of the program is not emphasized here, only external behavior is tested. Consequently, mostly functional testing is performed at these levels.

Further, testing object-oriented programs has different issues. There are several reasons that traditional testing cannot be directly applied to test object-oriented programs. Few of them among others are: classes cannot be tested directly, only an instance of a class can be tested. How can an abstract class be tested, as it cannot be instantiated? Lack of sequential control flow within a class requires different approaches of testing. State-based testing and incremental testing for subclasses etc are testing methods for object oriented programs. State-based testing is a technique to test whether or not the methods of a class interact correctly among themselves by monitoring the data members of the class. It is to test all the methods of a class, one by one, against the set of states that the object can take. During incremental testing class hierarchies are tested.

1.2 SOFTWARE REUSE

Systematic, formal 'reuse' term was proposed at the NATO software Engineering Conference in 1968 [3]. Software development involves understanding the requirements followed by designing (identifying components and their integration) and implementation. It is always understood that something that has been worked out earlier and is usable (as specification or component) can be reused to bring down the cost of development and possibly minimize the failures (as time tested entities are always better than a new item that has yet to be thoroughly tested).

The idea is to store such entities that have been developed and are of use in applications later. The general understanding is that almost all the applications in a domain may have many common components and such components can be developed once and reused many times. Similarly various types of specifications may also be reusable. Various methods of

review/analysis and verification may also similarly be reusable. Different architectural/design experiences may be reusable across applications and across domains.
This clearly shows that the scope of reuse is vast. Software is related to two issues: one "developing generic software components that can be reused in many applications" and other one "development with these reusable software components". Software reuse can be classified in various ways according to development scope, modification, approach, domain scope, technologies used, management and reused entity, as given in the Figure 1 [4].

Development Scope	Modification	Approach	Domain scope	Management	Reused entity
Internal	White box	General	Vertical	Ingrained	Requirements
External	Black box	Compositional	Horizontal	Planned	Design
	Adaptive	In-the-small		Coordinated	Component
		In-the-large		Monitored	Architecture
		Indirect		Initial	Test cases
		Direct			
		Carried over			
		Leveraged			
		Loose			
		Tight			

Figure 1: Classification of Software Reuse

Design patterns and frameworks are two technologies that provide high-level of reuse. Design patterns solve specific design problems and make object oriented design more flexible, elegant and ultimately reusable. *A design pattern describes solution to a recurring problem in software design phase* [5]. Frameworks are one of the object-oriented reuse techniques that provide highest level of reuse. It provides the reuse of the whole architecture, design and implementation of common features in all applications belonging to a particular domain. A framework is the skeleton of an application that can be customized by an application developer [6].

2 REUSE POSSIBILITIES IN TESTING PROCESS

As said earlier, to get the maximum benefit of reuse, it should ideally be implemented on all the phases of software development rather than only at coding. It is evident that the total automation of whole testing process is not possible. The test case generation, the process of identifying a set of test cases that satisfy a test case adequacy criterion is known to be un-decidable [7] and hence human interaction throughout the testing process is a necessity. The analysis of test deliverables and debugging also require human interaction. In such a situation an environment, which automates the activities that can be performed by the software on the computer and enables the intervention by the tester whenever and wherever required, may be beneficial. Further, test design patterns may also be developed for different activities carried out during testing. In the following sub-sections, we tried to seek the possibility of reuse in different activities of software testing process.

2.1 REUSE IN TEST PLAN GENERATION

In general, test plan generation is the 1st activity in software testing. Although testing is done at the end of software development process but it should be planed at the beginning. A test plan is a document that defines the scope, approach to be taken (test criterion), and the schedule of testing as well as the test items for the entire testing process and the personnel responsible for different activities of testing [2]. Although this activity can not be automated but some of the activities like checking consistency between schedule defined in project plan and testing schedule, selecting test units from the detailed design (during unit testing) and the system design (during integration testing) can be automated. However, unit suggested by tools may not be good enough in terms of its testability, but at unit testing level tools for this can be defined once and reused many times. Further, if history of testing of the previously developed and tested projects are stored somewhere and accessible, it will be easy to decide which approaches should be used during each testing activity based on previous experiences. Features to be tested are also determined during test plan generation. Using history of previously tested applications of similar type can also reduce effort needed for this activity. Schedule specifies the amount of time and effort to be spent on different activities of testing. For novice testers guessing it correctly is difficult. By using experience of experienced testers they can guess it somewhat correctly. Personnel allocation activity identifies the persons responsible for performing the different activities. By reusing previous test plan for similar projects, the experts of different testing activities can be determined easily and reduce the risks of not performing testing well. The idea is, if test plan generation information of similar type of software, i.e. for applications belonging to same domain, is stored in library then during testing of any software belonging to that domain, testers can make use of this information and much of the time can be saved. Further, it would lead to better testing. As design patterns are developed and stored for design reuse, test plan generation patterns can be generated and stored that would have several decisions related to test plan generation and thus can be used millions of time without doing it again and again. Further, test plan templates should be developed for different domains that would help in developing test plan during testing of applications belonging to that domain. Templates for test plan generation may guide the novice testers so that they would feel confident that they are doing correct things and not leaving something necessary, by mistake.

2.2 REUSE IN TEST CASE GENERATION

Test case generation activity starts with deciding the testing criterion. According to that test criterion test cases are developed. It is difficult to decide whether a set of test cases satisfy a criterion or not. For structural testing, several tools are used for this purpose that provides feedback regarding what needs to be tested to fully satisfy the criterion. Test case generation cannot be fully automated. It is an NP hard problem. Thus to reuse test cases the only and best method is to use patterns for test case generation. These test case patterns describe for what type of functionality this would help to generate test cases. In this way, time involved in test case generation (the most difficult activity in testing process) can be reduced to some extent. Generally testers select test cases in an iterative manner starting with an initial test case set and selecting more test cases based on experience during execution of those test cases [2]. This matured set of test cases should be stored in the library for public access so that testers testing similar applications can get idea from this set and can design test cases easily, in less time. Regression testing is one of the examples of testing where test cases are reused "as-is". It is the selective retesting of a software system that has been modified to ensure that any bugs have been fixed and that no other previously working functions have failed as a result of the reparations and that newly added features have not created problems with previous versions of the software. We can say, it a reuse in project level. For applying reuse at domain level test cases should be developed for a domain so that one can reuse those test cases in the testing of the similar type of applications belonging to that domain. Although, test cases "as-is" reuse is possible only either in regression testing or when any unit is being used in some system development. For the later case, all the test cases would be used to see whether the integration of this unit with others make a negative effect on this unit. That is, with other units this unit is not performing as for what it was designed.

Test cases can be described in UML. UML has been standardized by OMG (Object Management Group) that is an industry-standard analysis and design notation. One should use it in creating test cases so that one can easily reuse them due to standardization. Further, generation of test cases for software design using UML would be easy. It may be easier to test some features, without executing the software, by seeing UML design of software if UML is used for this purpose. Further, its automation may also be possible.

2.3 REUSE IN TEST CASE EXECUTION AND ANALYSIS

With the specification of test cases, the next step in testing process is to execute them. Executing the test cases may require construction of deriver modules or stubs. A stub is a dummy routine that simulates a module. It is used in top down testing strategy where testing is started by testing

the top of the hierarchy, and modules are added incrementally that it calls and then test the new combined system. Drivers are needed to perform bottom up testing that starts from the bottom of the hierarchy. Thus derivers are needed to set up the appropriate environment and invoke the module, which has no subordinate. These stubs and derivers can also be developed already by identifying them by analyzing the domain and can be reused. For analysis of the execution results data is to be collected. Sometimes, *test procedure specification* document is used to execute the test cases [2]. This document specifies any special requirements that exist for setting the test environment, the methods and formats for reporting the results of testing and measurements, if needed, along with how to obtain them. Here is again the possibility of reuse of this specification document among applications belonging to the same domain. Data collection forms and software can be developed in a particular domain and reused. The process of test case execution involves instrumentation of probes in the source code, executing the source code and analyzing the probe values. Probes are instrumented in the source code to compute the coverage of testing, execution results that are produced in intermediate stages etc. The most common method of instrumentation is to insert some statements, along with probe variables, called probes in the program. Probe insertion can be done automatically by a preprocessor. The analysis of the probe values can also be done automatically by a postprocessor. These preprocessor and postprocessor can be reused if they are implemented as generic components and can be customized during use.
Different types of preprocessors/postprocessors can be constructed based on different kinds of probes: probes that are inserted where a path splits into multiple paths and where multiple paths merge or a different probe for each path. For a domain different preprocessors and postprocessors should be generated in advance to reduce testing cost. According to need, users may select the one among all. Test oracles are needed to test any program. A test oracle is a mechanism that can be used to check the correctness of the output of the program for the test cases. The output of the two is compared to determine if the program behaved correctly for the test cases. Test oracles should be automated to give always a correct answer. But it is very difficult to automate test oracles. Although, for some systems there exists automated oracles but since automation of oracles are done based on requirements specification it is not always possible to obtain an error-free oracle and hence automation of this activity cannot be taken up. Since there is a possibility of errors in specifications thus oracle would too produce wrong output. Systems for which oracles cannot be "automated", oracles created during testing of previous systems may be used for generating oracles for systems of similar types. Test oracles should be developed as components and put in repositories. These test oracles can be reused either "as-is" or after customization, if needed.

2.4 REUSE IN "ERROR CAUSES" DETECTION AND THEIR REMOVAL

During execution errors are detected. The next step is to locate the cause of errors and correct them. Here also, we have a possibility of reuse of the *test summary report* generated for a domain during testing of the application belonging to that domain. Test summary report stores the summary of the entire test case execution. The summary gives the total number of test cases executed, the number and nature of errors found, and a summary of any metrics data (e.g. effort) collected. This report can be reused either at project level or within domain. At the project level it is used to track the status of defects found during testing while, across the projects within a domain, it can be used to concentrate on errors specified in the report already.

2.5 REUSE IN EVALUATION/REVIEW OF TESTING PROCESS

The last step in testing process is to evaluate/review the testing carried out to determine its quality. The evaluation of testing process is necessary to see whether it has been carried out satisfactorily i.e. carried out thoroughly or not. As with many other verification methods, evaluation of quality of testing is done through "test review". And for any review, a formal document or work product is needed [2]. All the test deliverables are reviewed, using a formal review process, to make sure that the testing has been carried out with the policy specified in the test plan, test cases specification is generated for regression testing etc. In this activity, the formal review process is being reused. If the review will be done in an ad hoc way, that is, if there will not

be a formal (repeatable) process, testing evaluation may not be done successfully because testers would always try to show that testing is carried out successfully. The quantitative(s)/quantitative(s) that can be used to measure the quality of the scripts that are developed by Engineers can also be reused in a particular domain. The traditional qualitative(s)/quantitative(s) would be:
Religious practice of coding guidelines,
Duration taken for a test run,
Quality of documentation (Test script comments, Test script description document),
Classifying the Review Comments in to different levels (e.g. minor, major, trivial etc.) based on the impact of the same on the script execution,
And number of times the script is run on the test harness before the same is run successfully etc.

Thus we have seen that there is much possibility of reuse in testing process. For each activity carried out during testing, test design patterns can be created to store the decisions that should be taken during different situations. Test environment can be developed to aid testers. The test design patterns schema is described as follows [8]:

The Test Design Pattern Schema

Name
A word, or phrase, that identifies the pattern and suggests its general approach.
Intent
What kind of test suite does this pattern produce? This is a succinct statement, no more than two sentences.
Context
What test design problem does this pattern solve? When does this pattern apply?
Fault Model
What kind of faults does this pattern target and why will it hit them? This section must explain how the test suite will reach the targeted faults, how it can cause a failure to be triggered, and how a failure will be propagated to become observable.
Strategy
How the test suite is designed and implemented? A test strategy must address four issues.

- T*est Model* section defines a representation for the responsibilities and/or implementation that are the focus of test design.
- T*est Procedure* section defines an algorithm, technique, or heuristic by which test cases are produced from the model.
- *Oracle* section defines the algorithm, technique, or heuristic by which actual results to a test case are to be evaluated for pass/no pass.
- *Automation* section discusses automated approaches to test suite generation, test run execution, and test run evaluation. The strategy is typically presented by example.

Entry Criteria
What are the preconditions for design and/or running a test using this pattern?
Exit Criteria
What conditions must a test run achieved to met this pattern's test goals?
Consequences
What are the disadvantages and advantages of using this pattern? The costs, benefits, risks, and general considerations for using this pattern are discussed.
Known Uses
What are the known uses of this test design pattern? What are the efficiency and effectiveness of this pattern or similar strategies, as established by empirical studies?
Related Patterns
What other test design patterns are similar or complementary?

3 TEST ARTIFACTS AND TEST REUSE REPOSITORY

We have seen there are different artifacts that can be reused during testing. Two basic types of test knowledge are recognized: generic test knowledge, which applies to all projects, and project-specific test knowledge. The arrows in the figure 2 represent the possible kinds of test knowledge that may be placed in the test repository and have been already explained above.

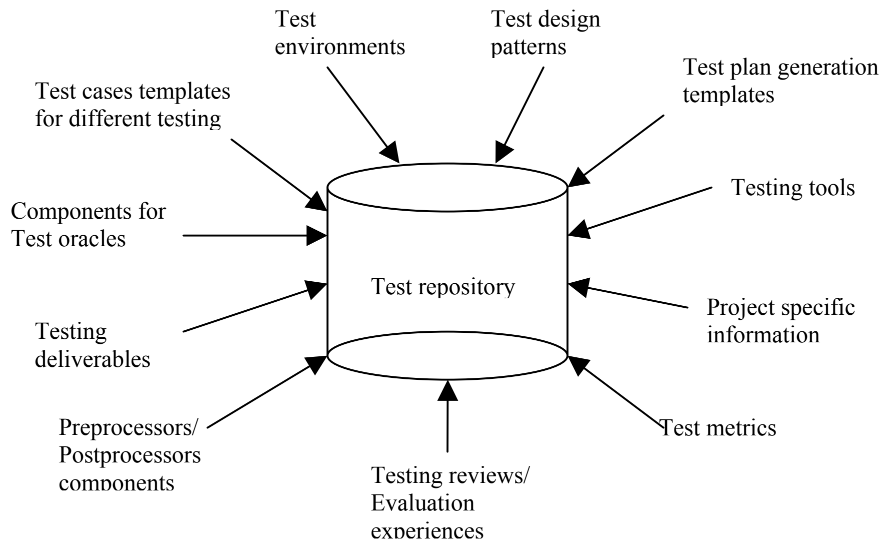
Test environments
Test design patterns
Test plan generation templates
Test cases templates for different testing
Testing tools
Components for Test oracles
Test repository
Project specific information
Testing deliverables
Preprocessors/ Postprocessors components
Testing reviews/ Evaluation experiences
Test metrics

Figure 2: The test repository

These artifacts must be stored in some reuse library in such a way that they can be accessible to users easily. Artifacts in test reuse library should be categorized in such a way that they can be accessed easily. Categorization of different testing artifacts is explained in figure 3. Some artifacts can be categorized based on domain while others may not. It is shown in column one. Other criteria for classifications are shown in column second. Second column for some artifacts is empty because they cannot be categorized except based on domain.

Figure 3: Kind of test artifacts

Testing artifacts	Categorization	Examples
Preprocessors/ postprocessors (According to domain)	According to the type of probes	1. A different probe is inserted for each path. 2. Inset a probe where a path splits into multiple paths and where multiple paths merge. 3. Inset a probe after each line of code etc.
Test cases (According to domain)	1. According to level of testing 2. According to "as-is" reusable versus customizable:	1. Unit testing, integration testing, system testing 2. Test cases (project level reuse), templates, test design patterns.
Testing tools	1. According to activities 2. According to particular testing	1. Test case evaluation tool, probes instrumentation tool etc. 2. Tools for data flow-based testing, mutation testing etc.
Test environments (According to domain)		Environment for unit testing, environment for test plan generation etc.
Test oracles (According to domain)	According to "as-is" reusable versus customizable:	Automated test oracles (test oracle components), test oracle templates
Testing metrics	According to activities	Defect removal efficiency, testability etc.

Testing deliverables (According to domain)		Test case specification report, test summary report, test log etc.

4 CONCLUSIONS

Many surveys on software testing and reuse have been carried out and it has been seen that reuse is not attempted in testing phase as in other phases of software development. Although some tools and test design patterns are being used in the field of software testing, but are not enough to say that reuse is being successfully applied in testing phase. In this paper, we have shown that there are various possibilities of reuse in testing process. Even, all the activities, carried out during software testing, have large potential for reuse. Although the whole testing process cannot be automated, testers can be aided through test design patterns, templates and test environments. We have highlighted the artifacts that can be reused during testing phase. These testing artifacts should be stored in reuse repository for access to testers. We have also explained the categorization scheme for different reusable test artifacts. This classification will help testers to locate and reuse these artifacts easily.

References

1. Tripathi A. K. and Gupta M., *Some Observations on Reuse Types, Technologies, Practices and Problems*, International Journal of Information and Computing Science, Vol.7, No.1, 2004.
2. Jalote P., “An Integrated Approach to SOFTWARE ENGINEERING”, ISBN 81-7319-271-5, Second Edition, Narosa Publishing House, 2000.
3. Smolarova M., Navrat P.,“ Software Reuse: Principles, Patterns, Prospects”. http://www.dcs.elf.stuba.sk/~/smolar, http://www.dcs.elf.stuba.sk/~/navrat.
4. Frakes W., Terry C., “Software Reuse and Reusability Metrics and Models”, Jan-1996.
5. Gamma E, Helm R., Johnson R. and Vlissides J., “DESIGN PATTERNS: Elements of Reusable Object-Oriented Software”, ISBN 0-201-45563-3, Addision Wesley, 1998.
6. Froehlich G., Kame1 A., Sorenson P., “Exploring O-O Framework Usage” In Proceedings of ICSE, 2000.
7. Rabins Porwal, Testing Classes in Object-Oriented Software: Issues, Analysis and Approaches, Ph.D. Thesis, DayalBagh Educational Institute, Agra, 2004.
8. RBSC Corporation provides *m*Verify test engineering services along with software engineering services. About this information can be seen on the following web site: http://www.rbsc.com/pages/TestDesignTemplate.htm.

Authors Biography

Miss Manjari Gupta: Ph.D. Research Scholar (and Teaching Assistant for Computer Sc at BHU), Dept. of Comp. Engg., IT-BHU, has done M.Sc. (Computer Science) from J.K. Institute of Applied Physics & Technology, Allahabad University. Her research area is Software Reuse. She is exploring the possibilities of reuse in software development processes to improve the quality of the development process and thus of the products/artifacts developed. Her work includes identifying frameworks and design patterns in testing field apart from object-oriented software development.

Meeta Prakash: obtained her Master degree and PhD in Computer Engineering and the dissertations at both the levels dealt with Software Testability. She has to her credit one publication in the International Journal of Information and Computing Science and one in Computer Society. She joined the Hexaware Software, Bangalore after Obtaining the PhD degree from Computer Engineering Department, Institute of Technology, BHU, Varanasi.