CHAPTER 7

SYSTEM TESTING

7.1 Introduction

This chapter explains testing done on DPDV system. Different kinds of testing have been done on the DPDV system that are unit testing, integration testing, system testing, and acceptance testing. And each of them will be explained here. Finally this chapter explains the evaluation of the DPDV system.

7.2 Testing stages

The testing on DPDV system is carried out in different stages that are preparation of test data, unit testing, integration testing, system testing, and acceptance testing. Each one of them will be discussed in the following subsections.

7.2.1 Preparation of test data

It is difficult to find standard dataset or code samples to serve as test data for DPDV. According to Stelting and Maassen (2002), there is no reference system with well-documented patterns that is freely available for testing pattern search algorithm. Antoniol (2006) claims that no public domain object-oriented system has both code and documentation of design patterns. As a result, source code must be analysed manually to assess the retrieved pattern. Heuzeroth (2003) highlighted that no real world software is well documented. The human analyzer can detect patterns by hand checking to assure them as correct.

In this study, the preparation of test data includes two steps that are:

a) **Collecting some Java code samples:** Three sets of java code samples have been collected to test the DPDV. The first set (Set 1) is from the book of Stelting and Maassen (2002). This book is about design patterns in Java and one of the Sun Microsystems Press Java series. The second set (Set 2) is from the DPDV project supervisor. The third (Set 3) is from the website published by Truett (1998). The author of this website is prominent in the area of software architecture and specifically in design patterns.

b) Identifying the design pattern used in the code manually: The three sets of Java code samples collected are checked manually to identify the designs pattern used in the code. Table 7.1 shows the set of Java classes along with their design patterns used for Set 1 Java code sample, Table 7.2 shows the same for Set 2 Java code sample and Table 7.3 shows the same for Set 3 Java code samples respectively.

Table 7.1

Set 1 Java code samples (Stelting and Maassen, 2002)

Sample	Classes in the Java code sample	Pattern used
1	Chovnatlh, ChovnatlhImpl, Contact, ContactAdapter, RunPattern	Adapter
2	BaseList, ListImpl, NumberedList, OrderedListImpl, OrnamentedList, RunPattern	Bridge
3	Contact, ContactImpl, Deliverable, Project, ProjectItem, Task	Composite
4	Contact, ContactImpl, Deliverable, DependentProjectItem, ProjectDecorator, ProjectItem, RunPattern, SupportedProjectItem, Task	Decorator
5	Currency, InternationalizationWizard, InternationalizedText, Nation, PhoneNumber, RunPattern	Facade
6	Address, AddressImpl, CleanState, Contact, ContactImpl, DirtyState, ManagedList, RunPattern,	Flyweight

	State, StateFactory	
7	Address, AddressBook, AddressBookImpl, AddressBookProxy, AddressImpl, DataCreator, FileLoader, RunPattern	Proxy

Table 7.2

Set 2 Java code samples

Sample	Classes in the Java code sample	Pattern used
1	Bulb, Fan, Switch	Null
2	CircleShape, DrawingAPI, DrawingAPI1, DrawingAPI2, Pattern, Shape	Bridge
3	Image, PExample, PImage, RealImage	Proxy
4	CGraphic, Ellipse, Graphic, Program	Composite
5	Pattern, UserfriendlyDate	Facade
6	DWindowTest, HorizontalScrollBarD, SimpleWindow, VerticalScrollBarD, Window, WindowD	Decorator
7	Factory, FDemo, Gazillion	Flyweight
8	DemoSquarePeg, RoundHole, SquarePeg, SquarePegA	Facade

Table 7.3

Set 3 Java code samples (Truett, 1998)

Sample	Classes in Java code sample	Pattern used
1	LooseLeafTea, TeaBag, TeaBall, TeaCup, TestTeaBagAdaptation	Adapter
2	CherrySodaImp, GrapeSodaImp, MediumSoda, OrangeSodaImp, Soda, SodaImp, SodaImpSingleton, SuperSizeSoda, TestBridge	Bridge
3	OneTeaBag, TeaBags, TestTeaBagsComposite, TinOfTeaBags	Composite
4	ChaiDecorator, Tea, TeaLeaves, TestChaiDecorator	Decorator
5	FacadeCuppaMaker, FacadeTeaBag, FacadeTeaCup, FacadeWater,	Facade

	TestFacade	
6	TeaFlavor, TeaFlavorFactory, TeaOrder, TeaOrderContext, TestFlyweight	Flyweight
7	PotOfTea, PotOfTeaInterface, PotOfTeaProxy, TestProxy	Proxy

7.2.2 Unit testing

White box testing is typically used for unit testing. According to (Gain, 2008), white box testing examines the internal procedural detail of system components**.** White box testing ensures that all statements and conditions have been executed at least once.

White Box testing checks the following (Gain, 2008):

1. All independent execution paths
2. All logical decisions on both true and false sides
3. All loops at their boundaries and within operational bounds
4. Internal data structures to ensure validity

White box testing was done to test "Structure" class, "FilesLoader" class, "Detector" class, "ClassDetails" class, "Visualizer" class, and "ReadMethods" class of DPDV and was done on every version of the DPDV system.

After doing the white box testing on first version of DPDV, it was discovered that the data structure ("ClassDetails" class) was not correct which resulted in wrong detection of patterns. Thus, in the second and third version, the data structure has been improved.

7.2.3 Integration testing

According to (Gain, 2008), black box testing is used for integration testing. Black Box testing examines fundamental interface aspects without regard to internal structure. Integration testing exploits high-level specifications.

Black Box testing checks the following (Gain, 2008):

1. Incorrect or missing functions
2. Interface errors
3. Errors in data structures or external database access
4. Behavior or performance errors
5. Initialization or termination errors

Integration testing has been done whenever a unit is added to the system to uncover integration errors. All the errors encountered at this stage were solved.

7.2.4 System testing

System testing was done to ensure the accuracy of the DPDV system through testing the three sets of Java code samples on the various DPDV versions. The final version proves that the system works fine and good enough to be used as a detection tool for structural design patterns.

This stage also involves matching/comparing the manually identified design patterns against the automatically detected design patterns in order to obtain the accuracy/detection rate.

The matching/comparing stage was done for the three different versions of DPDV. Table 7.4 shows the results for Set 1 sample code tested using the first version of DPDV and yielding an accuracy rate of “42.86%”. Out of those seven Java code samples, three code samples when loaded into DPDV produced results that matched with the manually identified design patterns, while four of the code samples produced different results.

Table 7.4

Testing Results of the DPDV Version 1

DPDV Version	Total testing code samples	Matching results	Un matching results	Accuracy/Detection rate (%)
1	Set 1 (7)	3	4	42.86%

The rationale behind this low detection rate was the data structure of the DPDV was not coded properly according to the rules of design patterns. Therefore, the data structure was improved in the second version of DPDV.

Besides, the second version of DPDV was tested with more code samples: Set 1 code samples and also Set 2 code samples. Table 7.5 shows the percentage of matching between the detection results produced by DPDV and the manually identified design patterns in the code samples.

Table 7.5

Testing Results of the DPDV Version 2

DPDV Version	Total testing code samples	Matching results	Un-matching results	Accuracy/Detection Rate (%)	Overall Rate (%)
2	Set 1 (7)	7	0	100 %	80 %
2	Set 2 (8)	5	3	62.5%	

The rationale behind this increase in the detection rate is mainly because the data structure has been improved and more variation of the rules were built into the DPDV system. Version 2 of DPDV showed better performance than Version 1.

In order to achieve higher detection rate, the detection rules were further refined in the third version of DPDV. This version is also tested with more code samples: Set 1 code samples, Set 2 code samples, and Set 3 code samples. Table 7.6 shows the percentage of matching between the detection results produced by DPDV and the manually identified design patterns in the code samples. This version showed increased detection rate when compared with the second version of DPDV.

Table 7.6

Testing Results of the DPDV Version 3

DPDV Version	Total testing code samples	Matching results	Un-matching results	Accuracy/Detection Rate (%)	Overall Rate (%)
3	Set 1 (7)	7	0	100 %	95.45 %
3	Set 2 (8)	8	0	100 %	
3	Set 3 (7)	6	1	85.71%	

Figure 7.1 shows the overall accuracy/detection rate for the three versions of the DPDV system. An overall accuracy/detection rate of 42.86 %, 80 %, and 95.45 % has been achieved for DPDV Version 1, Version 2, and Version 3 respectively.

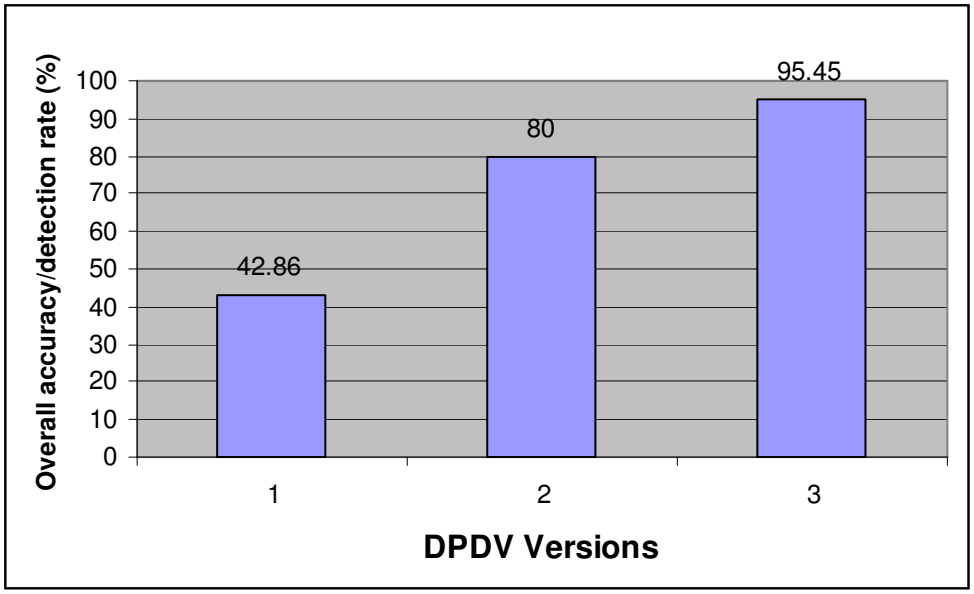

Figure 7.1: Overall accuracy/detection rate (%) for each DPDV version

7.2.5 Acceptance testing

Acceptance testing is done to ensure that the system requirements have been achieved and the quality of DPDV is acceptable. Five users had been asked to use the DPDV system.

The results of the acceptance testing done on first version and second version of the DPDV system was that the user interface of DPDV was not user-friendly. The DPDV first version used to load one file at a time. To solve this limitation, DPDV was modified to load any number of files at a time.

Moreover, the earlier version of DPDV used to show the detected design pattern by showing a pop-up text box message to a user. This type of detection results was difficult to justify the existence of the detected design pattern to the user. This problem was rectified by showing some visualization (in terms of UML class diagram) of the detected design patterns.

Besides, all the classes' details pertaining to the detected design patterns were also shown on a separated screen.

7.3 Evaluation of DPDV

The following table (Table 7.7) shows the evaluation of DPDV version 3 using the same criteria used in Section 2.6.3 to analyze other existing systems.

Table 7.7

Characteristics of DPDV system

Criteria	DPDV
Design Patterns that can be Detected	Structural design patterns
Programming language used to develop the system	Java
Samples Codes	3 sets of Java code – refer to section 7.2.1 for the details
Detection Method	DPDV accepts Java files as inputs from users. The tool uses the Reflection technique to extract the static and structural relationships between classes and objects that existed in the Java code. The tool then arranges the extracted information in a data structure defined as a class. After that, the tool examines the extracted information and compared them with the rules of each of the design patterns to determine the possible design pattern that existed in the Java code
Strengths	- The identified rules of each structural design patterns are built into DPDV system - This system provides users with design patterns representation preferences. The user can choose to view the detected design pattern textually or visually by showing UML class diagram. - Some functions of the DPDV system can be reused by other developers to enhance the DPDV or develop other systems which are similar to DPDV.
Weaknesses	- DPDV system is able to detect structural design patterns in code written in Java language only. However this can also be regarded as its strength since most existing systems detect pattern in code written using C++ language. - Each java class should not have any package name specified because the DPDV will specify its own package name for easy detection.
Precision rate	95%

In a nutshell, DPDV shows that it has very high precision rate of detecting design patterns (95%) compared to other exiting tools. Moreover, DPDV provides users with design patterns representation preferences. The user can choose to view the detected design pattern textually or visually by showing UML class diagram. In addition DPDV is detecting design pattern in Java code which is lacking in the market.

DPDV uses reflection technique to extract the attributes of each entity and the relationships among them from the code. Then, DPDV arranges this information in a data structure defined as a class. DPDV detection technique can detect all structural design patterns. In comparison with the other detection techniques reviewed in the literature, the following can be concluded: bit-vector algorithm uses the software metrics to detect Abstract Factory and Composite patterns only; negative search algorithm can detect all patterns but it does not address different implementation variants for a given design pattern; multi-stage reduction uses the Object-Oriented software metrics to detect adapter, proxy, composite, bridge, and decorator only with a precision rate of 55 %; the Static and Dynamic Analysis algorithm uses the abstract syntax tree (AST) to detect patterns and the precision rate is unknown.

7.4 Summary

This chapter examined testing on the DPDV system and showed how the system was improved from version 1 till version 3. It also showed the improvement made to the system in terms of its functionality and users interface as a result of the comments gathered during users acceptance testing.