

UNIVERSITY OF CALGARY

Testing Process for Portal Applications

by

Harpreet Bajwa

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE
DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA

December, 2005

© Harpreet Bajwa 2005

UNIVERSITY OF CALGARY
FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled "Testing Process for Portal Applications" submitted by Harpreet Bajwa in partial fulfilment of the requirements of the degree of Master of Science.

Supervisor, Dr. Frank Maurer
Department of Computer Science

Dr. Victoria Mitchell
Haskayne School of Business

Dr. Guenther Ruhe
Department of Computer Science

Date

Abstract

Application development using portal technology has grown rapidly as a key enterprise strategy for integration of different business processes, and content into a single unified front end. Despite the increase in portal application development, processes and practices such as testing that significantly impact quality are limited. A case study and a survey of portal developers, that evaluated state-of-the-art in portal application testing, revealed many open issues. These issues make comprehensive testing of portal applications difficult. In addition, these difficulties highlight requirements for unit and integration testing approaches as well as tool support for testing portal applications. This thesis proposes approaches that address these issues. The viability of the developed testing practices and tools is validated through a preliminary study conducted in the industry. The overall goal of this work is to present a set of practices to the industry for improving the testing process of portal applications.

Publications

Some content, ideas and figures from this thesis have appeared previously in the following peer-reviewed publications:

Bajwa, H., Xiong, W., and Maurer, F. (2005) **Evaluating Current Testing Processes of Web-Portal Applications**, *Proceedings of International Conference of Web Engineering (ICWE 2005)* LNCS, Volume 3579, Jul 2005, Pages 603 – 605

Xiong, W., Bajwa, H., and Maurer, F. (2005) **WIT: A Framework for In-container Testing of Web-Portal Applications**, *Proceedings of International Conference of Web Engineering (ICWE 2005)* LNCS, Volume 3579, Jul 2005, Pages 87 - 97

Acknowledgements

There were times during the course of this work, as exemplified by Paul Anderson's quote that, "*I have yet to see any problem, however complicated, which when you looked at it the right way, did not become still more complicated*". I extend the most sincere gratitude during some challenging times to my supervisor, Dr. Frank Maurer for guiding me in the right direction. Your suggestions were invaluable for this work to take its present shape. I have learnt much from you in research, and on more valuable aspects of life.

Carmen Zannier, my colleague I sincerely acknowledge your role and effort in editing this work. Your insightful remarks changed the face of this thesis. To Wenliang Xiong: your role as a friend and someone who laid the technical foundation of this work. Lawrence Liu, I am grateful to you for being there. You have always made me aware of different perspectives to intellectual problems. I thank you for the countless conversations when everything appeared well beyond me. Chris Mann, I thank you for teaching me discipline in research and fruitful discussions during the course of this work. Grigori Melnik, your energetic spirit brought laughter in stressful times and your astute questions on this work were helpful. Jamie McIlroy, I extend my sincere gratitude for your willingness and enthusiasm to collaborate on this work. Without your constructive comments and sharp insight, understanding this complex topic would not be possible.

I thank my parents for believing, that I was far from home doing something worthwhile. I am grateful to my dear brothers for supporting in my educational endeavors and encouraging me at every step. I owe much to my sister, Kanwal Chohan for her support and love throughout this journey.

To Valar and Bhavya: my roommates for tolerating my strange mood swings and listening to my crazy research woes. Bhavya, I acknowledge your

valuable input in editing this thesis. Sushma, I thank you for always being there when I needed you. Sukh Mahil, your sincere friendship and support was invaluable.

I thank my friends from India Gautami, Vrinda, Deepika and Kaajal for all the good times and being there, when I needed you all the most. Anand S, I am grateful to you, for listening to my cryptic talks during this time. Zohra and Iqbal, my friends, you have inspired and supported me through some hard times. Your friendship provides strength to my life. Shilpa, I thank you for the confiding talks. Anita, I rejoice that you are a part of my life. Your positive energy and intelligence has enriched my life in many ways in all these years. Rohit, your never failing, forgiving friendship and support was comforting during these last six months.

Janaki: without your role, many pieces of my life would have fallen apart. I will always cherish our all-nighter philosophical discussions on the world and our lives. You have inspired me beyond words to always excel in academic pursuits. Your true friendship and love has added strength to my being.

Sonia, my sister without your support, I would have never seen the end. You have always provided me a reason to come home and your undying cheerfulness was forever welcome.

Finally, I would like to acknowledge the funding support by Sandbox Systems and the Department of Computer Science.

Above all, I thank AID and its members for adding meaning and purpose to my life in these two years.

Dedication

To my mother, for instilling trust in me and teaching me early in life, that true learning was the key to freedom. I owe much to your unconditional love and infinite faith in everything, I have ever done.

Table of Contents

Abstract	iii
Publications	iv
Acknowledgements	v
Dedication	vii
Table of Contents.....	viii
List of Tables	xiii
List of Figures	xiv
Chapter 1. Introduction	1
1.1 Background.....	1
1.2 Research Motivation and Scope	2
1.3 Research Problems.....	4
1.4 Research Goals	5
1.5 Specific Research Questions.....	5
1.6 Key Terminology.....	6
1.7 Thesis Structure	7
Chapter 2. Portal Application Technology and Testing.....	9
2.1 Portal Server and Portal Components.....	9
2.1.1 Portal Server	10
2.1.2 Portlet and Portlet Container	11
2.2 Portlet Related Concepts.....	15
2.2.1 Comparing Portlets and Servlets	16

2.2.2 Portlet and Portal Applications.....	17
2.2.3 Portlet API and Services.....	18
2.2.4 Portlet Development Related Characteristics	18
2.3 Portal Technology Standards.....	19
2.4 Portal Application Deployment	20
2.5 Testing Techniques.....	21
2.5.1 Test Driven Development.....	22
2.5.2 Unit Testing with Mock Objects.....	23
2.5.3 Integration Unit Testing.....	25
2.5.4 Functional Unit Testing	26
2.5.5 Performance, Stress and Security Testing	27
2.5.6 Web Application - Model, View and Controller Testing	28
2.6 Summary.....	30
Chapter 3. Portal Application Testing: Case Study	32
3.1 Objectives	32
3.2 Case Study Methodology Overview	33
3.3 Study Context and Participants.....	33
3.4 Data Collection and Analysis	34
3.5 Results.....	34
3.5.1 Testing Practices in the Company	34
3.5.2 Challenges in Testing Portal Applications	37
3.6 Case Study Limitation	39
3.7 Summary.....	39

Chapter 4. Portal Application Testing: Survey	40
4.1 Survey Methodology Overview.....	40
4.2 Survey Design and Sample Selection.....	41
4.3 Response Rate.....	43
4.4 Participant Demographics.....	43
4.5 Portal Server Deployment Infrastructure.....	46
4.6 Results.....	46
4.6.1 Unit Testing.....	47
4.6.2 Functional Testing.....	50
4.6.3 Performance and Load Testing.....	50
4.6.4 Portal Application Deployment.....	51
4.6.5 Challenges in Testing Portal Application.....	52
4.6.6 Interpretation.....	55
4.6.7 Design and Testability of Portlet applications.....	55
4.6.8 Deployment Process and Environment Complexity.....	55
4.7 Limitations of the Study Methodology.....	57
4.8 Summary.....	58
Chapter 5. Portal Application Testing Process.....	59
5.1 Requirements for Portlet Testing.....	59
5.2 In-Container Testing of Portlets.....	61
5.2.1 WIT Testing Framework.....	62
5.2.2 Usage Scenario of WIT.....	63
5.3 Portlet Testing using Mock Objects.....	73

5.4 Portal Application Testing Process.....	74
5.4.1 Unit Test Environment Level Tests.....	74
5.4.2 Staging Environment Level Tests.....	76
5.4.3 Production Environment Level Tests	78
5.5 Summary.....	78
Chapter 6. Empirical Evaluation	80
6.1 Selection of the Methodology.....	80
6.2 Objectives	80
6.3 Study Methodology and Participants.....	81
6.4 Results.....	82
6.5 Interpretation.....	89
6.6 Anecdotal Evidence	91
6.7 Summary.....	92
Chapter 7. Conclusions	93
7.1 Research Problems.....	93
7.2 Thesis Contributions.....	93
7.3 Future Work and Conclusion.....	95
References.....	97
Appendix A. Portal Technology	103
A.1 Review of Portal Server Framework Services.....	103
A.2 Portlet API	106
Appendix B. Testing Tool Evaluation.....	107
Appendix C. Survey Materials	109

C.1 Survey Introduction Form.....	109
C.2 Survey Questionnaire.....	110
Appendix D. Ethics Approval.....	114
Appendix E. Perceptive Study Materials.....	115
E.1 Pilot Study at U of C	115
E.2 Consent Forms.....	118
E.3 Post Study Questionnaire	120
Appendix F. Co-Author Permission.....	122
Appendix G. Materials and Raw Data.....	124
Glossary	125

List of Tables

Table 2-1: Key differences between Portlets and Servlets.....	16
Table 3-1: Portlet Application Characteristics.....	36
Table 4-1: Type of Portal Server Environment Used.....	46
Table 4-2: Survey responses indicating the testing techniques needed.	53
Table 6-1: Perceived Usefulness of WIT.....	83
Table 6-2: Perceived Usability of WIT.....	84
Table 6-3: Who should write and Execute ICT Tests.....	87
Table 6-4: Type of Portal Sever Environment for Running ICT Tests.....	88

List of Figures

Figure 1-1: Scope of Research -the inner shaded box represents the scope of this work.....	4
Figure 2-1: Portlets aggregated to form personalised Yahoo public portal page.	12
Figure 2-2: Portal Server Components and Architecture (Wege and Chrysler, 2002; Hepper, 2003).....	14
Figure 2-3: Model, View, and Controller Architecture for Portal Applications.	19
Figure 2-4 Testing techniques and scope of testing Web application components adapted from (Massol and Husted, 2003).	22
Figure 2-5: An overview of testing techniques for Model, Controller and View layers.	29
Figure 3-1: State-of-the-testing practices at the company according to model, view and controller layers.	35
Figure 3-2 Case study- number of methods versus test methods in the portlet application.	37
Figure 4-1: Web Application Development Experience of Survey Participants.	43
Figure 4-2: Portal Technology Experience of Survey Participants.....	44
Figure 4-3: Survey results showing automated versus manual Testing.	47
Figure 4-4: Survey results showing how unit test cases are executed.	48
Figure 5-1: Shipping Portal Application in View Mode- Account Details Portlet.	64
Figure 5-2: doView Method - Accounts Portlet Class.	65

Figure 5-3: doView Test Case - Accounts Portlet Test Class.....	67
Figure 5-4: actionPerformed Method - Accounts Portlet Class.....	69
Figure 5-5: actionPerformed Test Case - Accounts Portlet Test Class.....	70
Figure 5-6: Credential Vault Portal Server Service.....	71
Figure 5-7: Snippet of WIT Test Configuration XML File.....	72
Figure 5-8: Test Execution Results using WIT.....	73
Figure 5-9: Portal Application Testing Process- testing activities in different run-time environments.....	76
Figure 6-1: Responses showing likely future Usage of in-container testing approach using WIT.....	85
Figure 6-2: Responses showing likely Future Usage of Mock Objects.....	86
Figure 6-3: WIT as the top Feature of the week on the Portlet Community Website.....	92

Chapter 1. Introduction

This thesis describes testing of portal applications by outlining the testing process using novel testing techniques and tools. To this end, existing testing techniques in use were evaluated by conducting a case study and a survey. This chapter begins with a brief introduction of portal applications and reasons for their significant growth as an enterprise technology. Next, I discuss the value of testing and limitations of existing testing practices building the motivation for this research. Then, I identify the scope of research work in the area of testing. Finally, research problems and goals of this work are highlighted. I conclude with an organizational overview of this thesis.

1.1 Background

Portals are Web based applications that provide a single integrated point of access to personalised information, processes and applications. In addition, they integrate diverse interaction channels and information at a central point providing an aggregated view to the user. These applications are developed using Web application technology, for example Java 2 Platform Enterprise Edition (J2EE).

Enterprises are increasingly seeking to aggregate heterogeneous backend functionality and business processes into a single integrated point of access to information. As a result, enterprise portals have found their way into the mainstream business world. Shilakes and Tylman (1998) coined the term enterprise portal. They define “enterprise portals as applications that provide an amalgamation of software applications that consolidate, manage, analyze and distribute information within and outside of an enterprise.” These applications also include business intelligence, content management, data warehouse and data management functionality which is accessible using a single

gateway. Results of research conducted by financial analysts (Kastel, 2003) show that most enterprises intend to implement a portal solution within a short period of time, further augmenting the growing importance of enterprise portals. The significant growth of portal technology is attributed specifically to solving enterprise level challenges of integrating business processes and data. Lack of such an integration results in the end user interacting with multiple, inconsistent user interfaces to get a single task completed. In addition, to providing a consistent user experience, portals have the ability to integrate different media sources and render information in different formats for various devices.

1.2 Research Motivation and Scope

A lack of rigor in testing portals may significantly impact quality as enterprise systems become integrated using portal technology. Consequently, portals may fail to deliver expected functionality to the end user. Therefore, it becomes essential to establish a testing process with solid foundations, and identify practices that will promote effective testing of portal applications. A survey conducted by the Cutter consortium (2000) reported top problem areas of large scale projects in the Web application domain as failure to meet business needs (84%), lack of required functionality (53%) and poor quality of deliverables (52%). One possible reason for this was attributed to inadequate testing of these applications.

So far, most of the efforts from the portal community are focused on the technical and technological issues of portals. For example, proposals have been documented in the area of best practices for designing and developing applications using portal technology (Hepper and Lamb, 2004). However, as portal technology is used effectively in enterprise environments the focus will change towards the quality aspect of the portal applications. There are no recommendations or guidelines for the testing process of portals. No in-depth studies have been undertaken to evaluate how portals are being tested in the industry. The thesis work conducted helps to fill this gap. A prerequisite to providing support for better tested portal applications is an early assessment of the existing testing process, and the

nature of challenges that restrict comprehensive testing in the industry. I analyzed problems with the testing process for portals reported by our industry partner (Sandbox, 2005). As a follow-up to this, a survey of testing practices was conducted with experienced portal developers. The survey focused on testing processes, testing techniques and automated testing tools used in the industry. Open issues in testing of portal applications were revealed, which I have investigated and addressed as part of this work.

The scope of this work (Figure 1.1) is confined to the area of testing. Starting at the outermost box of Figure 1.1, *software engineering* looks at processes and tools for developing high quality software applications (Abran et al., 2001). *Web Engineering* is a discipline within software engineering that addresses the need for systematic techniques, tools and guidelines for design, development, test and maintenance of Web based applications (IEEE Multimedia). A portal application is a specific type of Web application with special characteristics that make it necessary to develop new methods for an effective testing process. This work investigates testing techniques and tools to perform portal application testing in a systematic manner. Specifically, I focus on portal applications based on the J2EE technology. However, there are other portal technologies such as Microsoft Sharepoint Portals.

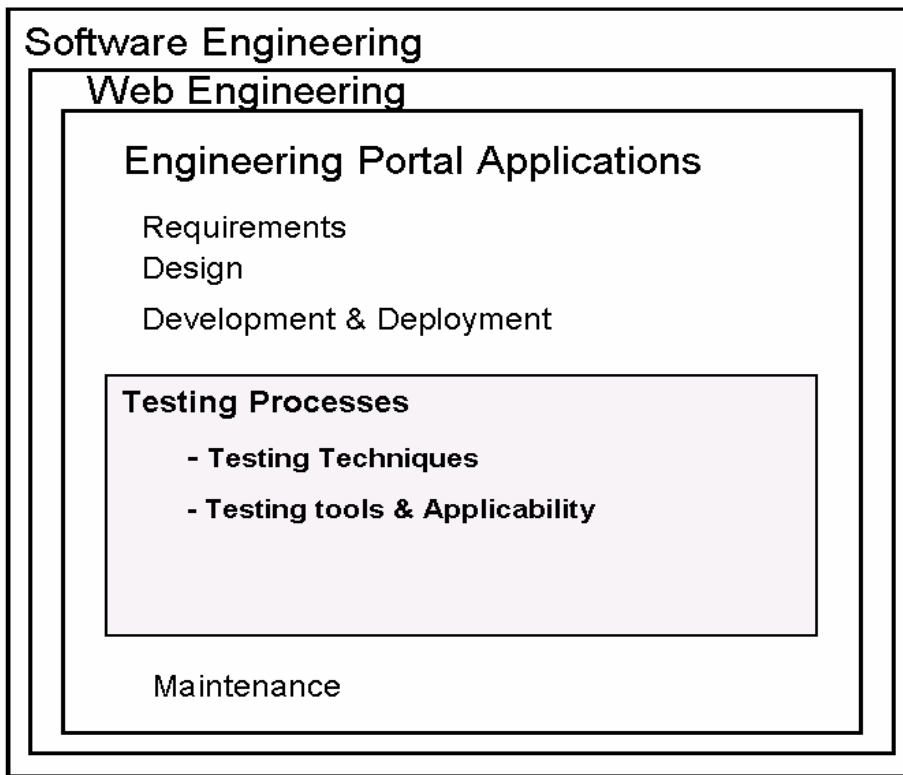


Figure 1-1: Scope of Research - the inner shaded box represents the scope of this work.

1.3 Research Problems

I will address the following problems in the thesis:

1. It is not known how a portal application is being tested in the industry (state-of-the-practice) and what difficulties exist that hinder automated testing of portal applications.
2. It is not known what testing tools and techniques exist that are appropriate for testing a portal application; if there is a need to extend these techniques and develop practices for portal application testing process.

1.4 Research Goals

In this thesis, I will address the problems mentioned above with the following goals:

1. I will empirically investigate and evaluate the existing process of testing portal applications to understand the testing practices in use and challenges in testing of portal applications. I will define and run an explorative in-depth case study and a broad survey to build this initial knowledge (*Problem 1*).
2. Based on the literature review and results from the study and survey in goal 1, I will develop a set of testing techniques. Using these techniques, I will describe how portal application testing process should be accomplished (*Problem 2*).
3. I will perform empirical evaluation to validate the testing techniques suggested in goal 2 (*Problem 2*).

1.5 Specific Research Questions

Outlining the testing process for portal applications (*goal 2*) is broken down further into following questions for establishing the right sequence of testing activities - testing processes:

1. How should testing be done (manual or automated)?
2. Who should perform testing?
3. Which components or collection of components need to be tested?
4. What techniques and tools can be used?
5. What is the scope of tests (unit, integration)?

1.6 Key Terminology

The concept of **software quality** though subjective is defined in terms of three factors adapted from (Culbertson et al., 2001) few failures in the field (lesser bugs), high reliability (seldom crashes) and customer satisfaction. The goal of any software testing activity is verification and validation (V&V). Schulmeyer (2000) defines *Verification* as the assurance that the products of a particular phase in the development process are consistent with the requirements of that phase and the preceding phase.

The term **process** in the context of this work is used to emphasize that testing is an orderly sequence of planned activities relying on well defined test strategies. **Testing** is a crucial activity in the software development process whose main goal is to reveal bugs (IEEE, 2002). The overall objective of testing is to improve the quality of the applications and end user experience.

Testing is an activity that is divided into several levels and phases analogous to the application development process. Terms related to software testing phases and techniques explained below are adapted from the Object-oriented (OO) (Binder, 2000) and Web testing literature (Kaner et al., 2002; Nguyen et al., 2003).

- **Unit testing:** This testing focuses on each program unit to ensure that the algorithmic aspects of individual units are correctly implemented. The goal of unit testing is to identify faults related to logic and implementation in each unit.
- **Integration testing:** After each unit is tested, the integration testing phase begins to test that an application built from individual units works correctly as a whole. The goal of integration testing is to identify whether a unit is adversely affected by the presence of another unit.
- **System testing:** After integrating software, system testing is performed to ensure that elements that are part of the system (for instance hardware and database) are adequately combined and the functional performance is obtained.

In order to design methods for testing in various phases and levels established above, testing techniques are developed. They test different aspects of an application and form the basic mechanism to assess the quality of it.

- **Functional or Black-box testing:** This form of testing treats the system as a "black-box", without any explicit knowledge of the internal structure and uses only the external structure that is visible. Black-box test design focuses on testing functional requirements without any concern for its implementation. This form of testing validates expected behaviour of the application from a user's point of view and can be associated with acceptance testing. Synonyms for black box testing include specification or behavioural testing.
- **White-box testing:** This form of testing allows the tester to focus specifically on using internal structural knowledge of the software to guide testing. This technique is complimentary to black box testing.
- **Gray-Box testing:** This form of testing incorporates elements of both white box and black box testing since it consists of methods derived from the knowledge of application and its environment. This technique is integral to Web application testing. As Web application comprise of numerous components (software and hardware) that must be tested in the context of system design to evaluate their functionality. (Nguyen, Johnson et al., 2003) define gray-box testing as *“using inferred or incomplete structural or design information to expand or focus black box testing”*.

1.7 Thesis Structure

This thesis is divided into seven chapters:

In chapter 2, I present an introduction to portal application technology, its key components and features that make these applications unique. Next, I discuss existing research on Web application testing by reviewing existing testing techniques and tools. The

chapter concludes with limitations of the testing techniques with reference to portal applications.

In chapter 3, I discuss the methodology for the explorative case study conducted at a company to understand the testing practices (Goal 1). Next, the results of the study are discussed. The chapter concludes with a brief summary of the study findings.

In chapter 4, I discuss the design of the survey conducted to provide broader insight into testing practices for portal applications (*Goal 1*). Next, the results of the survey are presented which includes the description of the empirical assessment and its findings. The chapter concludes with limitations of the case study and survey methods used.

In chapter 5, I summarise the challenges in testing. Next, I use this research and empirical results discussed in chapter 4 to develop a set of testing approaches and techniques for integrating them into the development process (*Goal 2*).

In chapter 6, I discuss the perceived benefits of this proposed process through a qualitative evaluation of the recommended testing approaches developed in chapter 5. This involves conducting a less formal evaluation of the suggested techniques in industry. Next, I use the research results to develop a set of practices which can provide enterprises with a direction in improvement of their own testing processes for portal applications (*Goal 3*).

In chapter 7, I conclude the thesis by summarising how I achieved each of the research goals. I also list my research contributions and suggest future work in the area of portal testing.

Chapter 2. Portal Application Technology and Testing

This chapter begins with an introduction to portal application technology, its components and execution environment to provide sufficient background knowledge of portal architecture and understand the characteristics that differentiate them from conventional Web applications. Next, a summary of the standards in place for portal technology is discussed followed by a brief overview of development and deployment processes of portal applications. Then, existing research on testing techniques for Web application is reviewed. I conclude with a summary of the testing processes and its limitations with respect to portal applications building the motivation for this research.

2.1 Portal Server and Portal Components

Support for portal technology evolved from the need to aggregate and render multiple streams of dynamic content to present it in a unified manner to the end user. As a result proprietary portal frameworks (Appendix A.1) have emerged providing portal server components to customers for building portal Websites and its extensions. In the context of this thesis the term “*component*” will be used in two ways. First, it is used in a manner similar to component based systems (Cechich et al., 2003) to describe any piece of software that provides services through its well defined interfaces with emphasis on the “*black box*” nature of components. Second, I extend its use in a way that describes pieces of application code with available source code that uses services provided by the black box components. Terms related to portals referenced throughout this chapter are explained in the glossary.

2.1.1 Portal Server

The Portal server provides basic infrastructure support for developing and deploying portals. In other words, it provides an environment for executing and managing portal applications. Portal Server offers services such as:

- **Content Aggregation:** A portal server gathers content from different sources for different users. It is responsible for aggregating content produced by individual portlets into a single portal page
- **Personalisation and Customisation:** A portal server is enabled to recognize different users and offer them content specifically configured to their needs. This service is based on gathering information about user communities and delivering customized content.
- **Single Sign On (SSO):** A portal server is an entry to a wide range of back end applications. It supports an authentication mechanism that does not require user authentication each time. The end user authenticates once and has unrestricted access to all the applications.
- **Content Management:** Portals gather content from different sources by implementing a syndication service that talks to every attached back-end system via an appropriate protocol. Built in support is also provided for standardized content formats, such as rich site summary, news industry text format (NITF) etc.
- **Collaboration:** These services are provided by a few portal frameworks (for example WPS) through a set of pre-defined portlets that allow for team-room chat, e-mail, sharing calendars and many other collaborative technologies.
- **Multidevice support:** Portals can prepare content for different interaction channels, such as those for wired and wireless phones, pagers, and faxes, by considering their characteristic capabilities.

The nature of services provided by a portal server is vendor specific. A review of different open source and commercial portal server run times is reported in Appendix A.1.

The rationale of reviewing the portal frameworks is to describe briefly the range of services provided. The report highlights various levels of service provided by products. They range from complete enterprise portal solutions to simple libraries for developing and running portlets. My work is focused more specifically on testing techniques using J2EE based IBM Websphere portal technology that provides the framework and runtime environment to build portal components and its extensions (PortalZone).

2.1.2 Portlet and Portlet Container

J2EE-based portals aggregate multiple applications into a single unified front end by integrating individual **components called portlets**. Portlets are described by (Hepper, 2003) as “*user facing applications that generate dynamic content from multiple backend sources*”. Thus, portlets are core application building blocks of portals. Portlets are deployed in a special environment called the **portlet container**.



Figure 2-1: Portlets aggregated to form personalised Yahoo public portal page.

Figure 2-1 shows a typical layout of a portal page divided into a number of independent areas. Each area is referred as a portlet. Each area contains information from a different source. Each portlet is a separate window that has different states such as normal, maximized, and minimized. In some cases the portlet acts as a mini Web browser containing html content obtained from another web server. Figure 2-1, shows a yahoo portal page with news and weather portlet windows that gather information feeds and announcements from another server. A portlet is defined as a Java technology based Web component, managed by a portlet container that processes requests and generates dynamic content. It is used by portals as pluggable user interface components providing a presentation layer to information systems. Portlet access can be restricted by assigning proper permissions based on the user roles they serve.

Portlet windows have standard modes (Hepper, 2003) (view, edit, help and custom) that indicates the current function portlet is performing:

- **View:** In the view mode, a portlet renders content fragments (small pieces of markup such as HTML, XML), display data, present content and provide core functionality to users.
- **Edit:** In the edit mode, the portlet provides the content and logic that allows the user to customize the behaviour of a portlet. This allows users to customise the information they want to see and how it should be presented to them.
- **Help:** In the help mode a portlet provides help screens that explain its purpose and expected usage.
- **Custom:** This mode is an optional feature that provides a specific piece of functionality depending on the purpose of the portal application.

A portlet container is the portlet runtime environment that provides portlet specific services. Typical responsibilities of a portlet container are to:

- Provide portlets with the required runtime environment and manage their life cycle. It is responsible for portlet instantiation, initialization, request processing and destruction
- Provide persistent storage mechanisms for *portlet preferences*
- Receive and execute requests from the portal server (described below) by retrieving content of the portlets it hosts.

Portlets and their runtime support (portlet container) are commonly featured technologies in many of the current portal building frameworks such as Apache (Jetspeed), IBM Websphere portal server (WPS) (WebspherePortal) and Oracle's Application Server Portal (Oracle).

The portal architecture is depicted in Figure 2-2 to explain components integrated with the portal server which is built as a servlet Web application. In addition Figure 2-2 explains how a client request is processed by the portal server using the services provided by the different components.

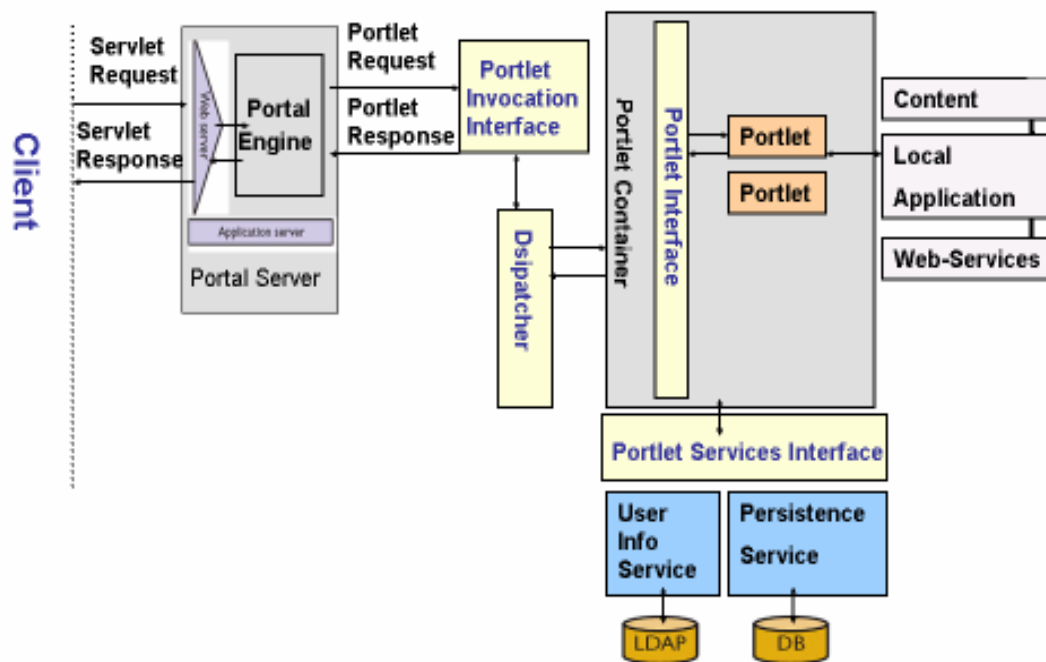


Figure 2-2: Portal Server Components and Architecture (Wege and Chrysler, 2002; Hepper, 2003)

A client request for a portal page is processed by the portal server through interactions with several components as shown in Figure 2-2. First, the portal engine implemented as a servlet application, receives a servlet request. Next, this request is transformed into a portlet request by the portal engine. Then, the portlet request is sent to the appropriate portlet via the portlet container, which is invoked from the portal server via the portlet Invocation interface as shown in Figure 2-2. Next, the request is processed by retrieving portlets for the specific portal page. The container invokes individual portlets to retrieve portlet content through the Portlet interface also highlighted in Figure 2-2. Finally,

the portal server aggregates multiple portlet responses returning them as a servlet response to the end user. The final portal page presented to the client represents an aggregate of several portlet windows which takes into account user preferences and device capabilities. One of the key responsibilities of the portal server is to receive and execute client requests for a portal page by retrieving and aggregating content of the portlets hosted by the portlet container.

2.2 Portlet Related Concepts

This section explains key concepts related to portlets. Identifying differences between portal applications and conventional Web applications is a prerequisite in answering whether it is possible to adapt existing testing techniques from the Web application domain to portals. Many aspects of portlet development are common to Web application development since the Portlet architecture is an extension of the Java Servlet architecture (which, in turn, is an extension of a Web server). However, unique aspects of a portal environment add complexity to the application model such as multiple portlets per page and portlet URL addressing. This section highlights these differences in detail.

2.2.1 Comparing Portlets and Servlets

Servlets and portlets are both Java-based Web components that generate dynamic content and are managed by specialized containers.

Table 2-1: Key differences between Portlets and Servlets.

Features	Portlets	Servlets
Runtime Environment and Component Administration	Communicate with the end user via the portal server and run in the context of a portlet container. They are administered dynamically for example they can be installed and removed while the portal server is running.	Communicate via the Web server and run in the context of a servlet container. These components cannot be installed and removed at runtime.
Content Generation	Content generated by portlets are mark up fragments that are aggregated to form a complete portal page by the portal server.	Servlets provide complete web pages although a servlet is a single piece of a much larger application.
Authentication Mechanism	Portlets operate in the context of a portal server responsible for validating user authenticity via SSO . Role based access of these resources is provided	Servlets are responsible for validating user authenticity. User authentication is a concern handled by application developers.
Request Processing	Process " doView " and " doEdit " requests, received from the portal server. Request processing is divided into an action phase for processing user actions and a render phase for producing the mark-up.	Process " doGet " and " doPost " requests, which map to HTTP get and post requests received directly from the Web browser.

URL Binding	Portlets are not directly bound to a single URL . Instead the portal page points to the page containing multiple portlets. Therefore, portlets cannot be called directly.	Servlets are bound to a single URL and can be called directly.
Response Predictability	Each portal page is an aggregation of several portlets. Multiple portlets can exist side by side with one another and each one may provide content and functionality different from the other. As a result, portlets maybe affected by the presence of other portlets providing lesser response predictability.	At any given time only a single servlet is fulfilling a user's request which provides a great deal of predictability . It can guarantee what is executed and returned to the client. Servlet can be sure that the returned content will not be affected by any other servlet.

Together with leveraging the complete functionality of servlets, portlets provide more specialized functions. Key differences between portlets and servlets are highlighted in Table 2-1. A portlet is built to support a well focused service rather than complex service which involves several business processes. This complex service is created by the portal server by aggregating portlet content. Web applications are perceived as self sufficient applications providing diverse functionality and a wide variety of contents (HTML pages hold a wide variety of contents) making them coarse grained. On the other hand, portlets provide a single piece of functionality making them fine grained.

2.2.2 Portlet and Portal Applications

A portlet application contains a group of related portlets that share a common context, for example images, property files and classes, and can exchange messages between each other. On the other hand, portal application comprises of multiple such portlet applications together with services used by these applications packaged together.

2.2.3 Portlet API and Services

The portlet application programming interface (API) provides interfaces (PortletResponse, PortletRequest, PortletContext etc) a portlet class can utilize. A portlet invocation request is handled by the service methods depending on the portlet mode requested by the client. Accordingly, the service methods (doView, doEdit, doHelp, doConfigure) associated with these modes is implemented as part of a portlet class.

Portlets depend heavily on services provided by the portal server environment. For example, services such as content access, search and location services. These services are provided as extensions to the Portlet API by plugging them into the portal server. Portlets access these services by querying the server for a specific service type and in return, receive an implementation of the service.

2.2.4 Portlet Development Related Characteristics

The Model-View-Controller (MVC) (in the Web context usually called the MVC model 2) design pattern (E.Gamma et al., 1995) is commonly used to achieve separation of responsibilities in a portlet application. This section describes the application of MVC to portlets (Hepper and Lamb, 2004). In a typical J2EE application, the types of components used are JSP, servlets, and Enterprise Java Beans (EJB). Figure 2-3, shows how the MVC architecture is applied while developing simple portal applications. Model in a portal application encapsulates business logic by retrieving data required to achieve a business function represented in J2EE by EJB, java beans, java classes.

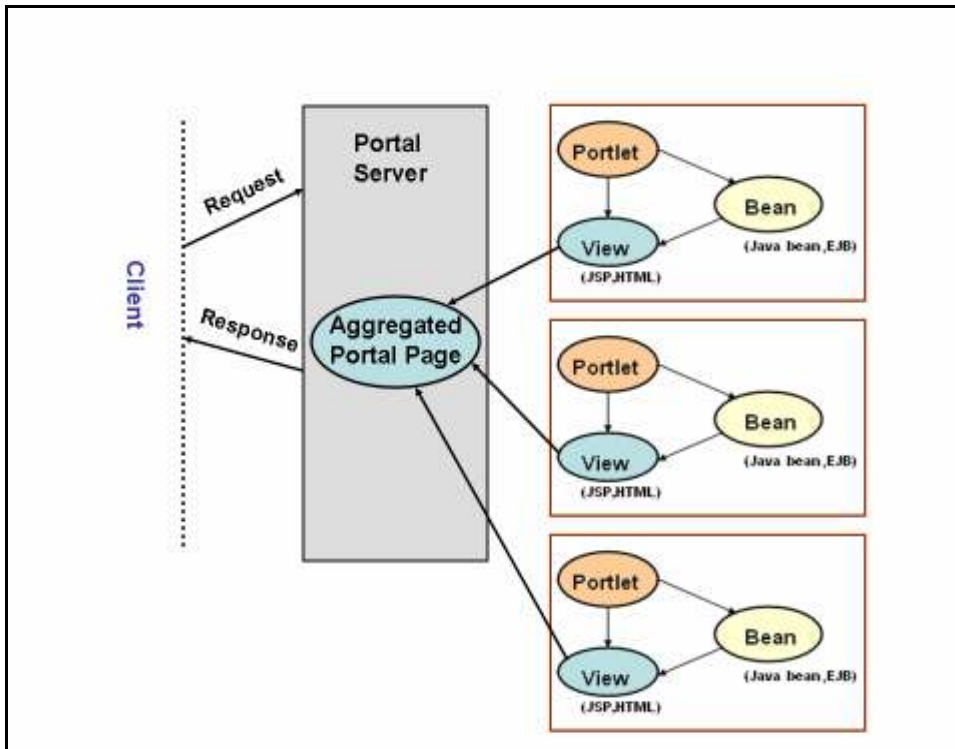


Figure 2-3: Model, View, and Controller Architecture for Portal Applications.

The portlet class is the controller with many functions. First, it evaluates the validity of the client request. Next, it determines the requested mode of portlet and accordingly executes the appropriate model component. Finally, it is responsible for invoking the correct view component.

The view component is responsible for rendering the presentation resource from the data returned by the model. The content returned by the view component is aggregated by the portal server as shown in Figure 2-3 to compose the complete portal page.

2.3 Portal Technology Standards

The emergence of an increasing number of enterprise portals has given rise to different vendor specific, mutually incompatible portlet APIs for application providers, portal customers, and portal server vendors. To overcome these problems and to enable **interoperability among portlets and portal servers** the Java Community Process (JCP)

has provided a standard of how portlets should be developed. This is described by the Java Specification Request (JSR168, 2003) that defines the API for standardizing the contract between the portlet and the portlet container. The goal of JSR 168, the Portlet Specification, is to enable interoperability between portlets and portals. The reference implementation of the specification is made available by (Pluto). The portlet API addresses areas of content aggregation, personalization, presentation, security and how the portlets should communicate with portlet container.

JSR 168 compliant portlets can be consumed as remote portlets using the Web Services for Remote Portlets (WSRP) protocol. WSRP is another important standardization initiative intended to simplify the creation of distributed applications by describing the communication protocol between portlet producer and consumer. The availability of these standards provides organisations deploying enterprise portals a range of standards-based portlet containers to choose from (write once and deploy on many platforms).

2.4 Portal Application Deployment

The portal application deployment process binds several portlet applications into the portal server environment. Each portlet application is packaged together with many portlets, deployment descriptor files and its resources into a Web Archive Portal Application file (WAR). This is followed by deploying the WAR target into the portal server run time environment.

Typical enterprises have three or more completely separate runtime environments for deploying portlet applications depending on different factors such as the size of the project and the quality processes in place. These run time environments have distinct purposes and will be referred in the remainder of the thesis:

Development and Unit Test Environment (UTE): This environment represents the local development environment used by the portal application developers to write, compile and unit test their code. The environment is the integrated development environment (IDE) that

allows developers to perform portlet application development, portlet testing and debugging in their local environments.

Test or Staging Environment: is the environment which is a close mirror of the production portal server environment where production critical tests are carried out. This is the first environment where the desktop application code is migrated to the runtime portal server environment. This is done to verify whether the application works before going live. Integration, functional and acceptance testing of the portal application code is performed in the staging environment before deploying the application for production use.

Production Environment: is the business production environment that has the full fledged portal server environment installed where the portlet application is deployed and released.

2.5 Testing Techniques

This section reviews the existing testing techniques for J2EE based applications. The figure below illustrates J2EE components and the scope of testing techniques represented by the red bars in the figure. It is important to note, that testing Web applications focuses on testing each component for example, unit testing business logic encapsulated in session beans. In addition, the components must be tested after they are integrated for example, database integration testing.

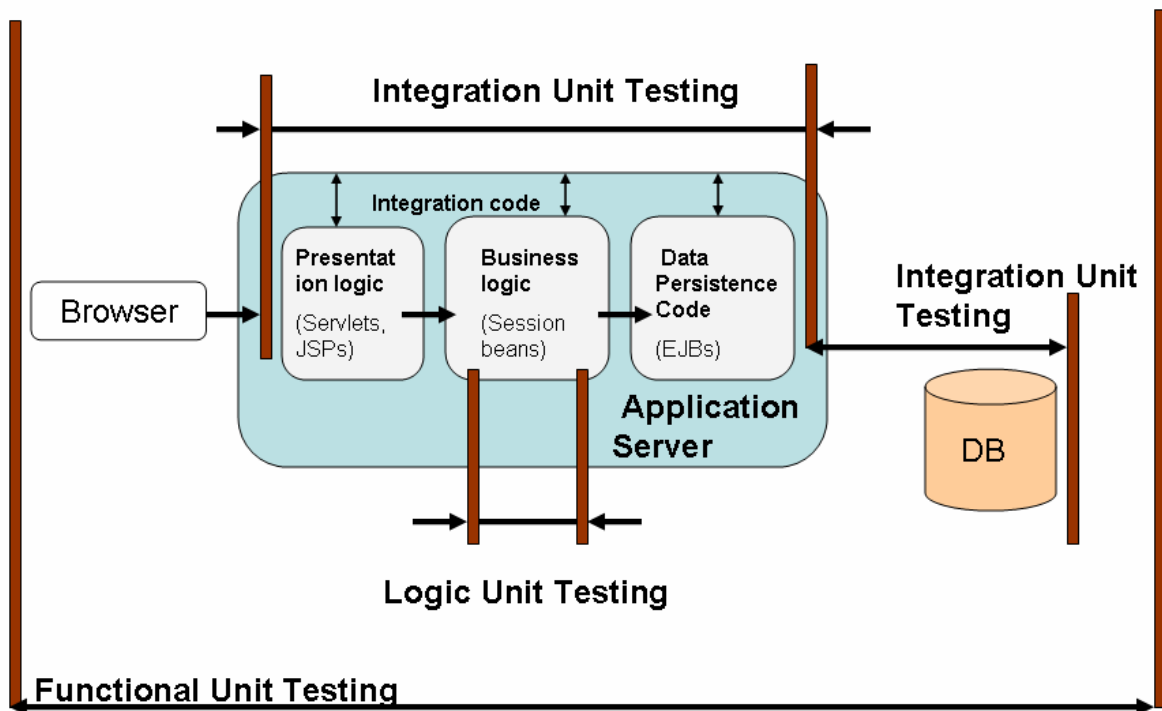


Figure 2-4 Testing techniques and scope of testing Web application components adapted from (Massol and Husted, 2003).

The limitations of the testing frameworks and tools for the testing techniques described are summarised in Appendix B.

2.5.1 Test Driven Development

Test driven development (TDD) (Astels, 2003) is a practice of agile methodologies such as Extreme Programming (XP) (Beck, 2000) that suggests :

- Tests are written first and allowed to fail before the functionality to pass the test is written. As a result, TDD (also called test driven design) shifts testing to the front of the software development cycle.
- Tests guide what functionality should be written. This allows developers to implement functionality just enough to make the tests pass.

- No code goes into production without associated tests. As a result, a suite of unit tests are created that form the basis for regression testing. This assures that adding new functionality does not break previously existing working code.

TDD is an **incremental development approach** using unit testing techniques (TDD and Unit-Testing). It focuses on verifying that units implemented by the developers work correctly. (JUnit) is the standard unit testing framework that automates unit testing of Java applications (Appendix B).

Tests are executed frequently (every few minutes) using TDD. The disadvantages of slow test execution in TDD (Smith and Meszaros, 2001) are two-fold:

1. **Increase in development time:** development time is increased considerably, if the test execution time increases. For example, assuming a developer runs the tests every 10 minutes while developing. Accordingly, they will run the tests eighteen times in a 3 hour programming session. A minute increase in test execution times increases development time by 10% (18 minutes). Therefore it is essential for tests to execute quickly likely in seconds for a typical test run.
2. **Delayed Testing:** If test execution is too slow, developers are more likely to defer testing resulting in delayed feedback. The impact of delay in testing is that identifying the change that may have caused an error after a series of changes is difficult. As a result, debugging becomes difficult and more time consuming.

Therefore, the effectiveness of the TDD approach is inversely proportional to the test execution time.

2.5.2 Unit Testing with Mock Objects

Mock Object testing (MockObjects, ; Mackinnon et al., 2000; Mackinnon et al., 2001) is a strategy to unit test methods that depend on interactions with other classes or the infrastructure. In essence, it provides minimal implementations of the services provided by the run time environment by using a simulated object called a Mock Object (MO). An

essential aspect of unit testing (Mackinnon, Freeman et al., 2001) is to test one feature at a time which is difficult if a unit test depends on a complex system state to be set up before the test executes. Mock Objects can reduce such problems. MO use simplifies the test structure and prevents the domain code pollution with testing infrastructure.

MO is a technique that supports TDD (Mackinnon, Freeman et al., 2001). Unit tests written with MO can be implemented using TDD first by writing the required simulated objects. This is followed by creating the set-up code using simulated objects for unit testing the intended functionality. Infrastructure choices can be deferred to a later time and developers can continue writing the application code without waiting on the choice and implementation.

Some benefits of using the MO approach are:

- **Reduced Test Execution Time:** Test execution time is reduced considerably by providing lightweight simulation of the dependencies. This is because the time taken to execute unit tests against MO versus a real object is less making it possible for developers to run tests frequently. This is important especially for TDD because TDD relies on tests to guide the development of the production code to ensure that code is working as required. The MO strategy, for instance, can be applied to servlet testing by using an API for simulating the servlet container provided objects (request, response, session context) making it possible to test the servlet logic without the overhead of testing in a real container. (ServletUnit) is a tool that supports a MO-based unit testing process of the servlet logic (Appendix B).
- **Rapid Test Feedback:** In order for tests to provide valuable and continuous feedback to the developers they should be executed frequently and quickly. Nevertheless, executing large number of unit tests against domain objects for example a real database may be slow to provide the rapid feedback developers need. The MO approach promotes faster test execution because unit tests run against simulated less complex domain objects.

As MO is different from the “*real*” objects that they replace, they do not assure that the methods under test will run correctly when deployed in the real production environment. They only allow for a finer grained unit testing of the business logic independent of the real context in which they run. Functional and integration testing against the “*real*” domain objects becomes important to ensure that the application works as expected. In some cases, domain objects can be hard to create to represent a complex external domain object and the effort to mock a complex domain object and maintaining its code is high. As a result, the benefit of creating MO may not be realised. On the contrary, the MO strategy may increase the test development effort.

2.5.3 Integration Unit Testing

Integration Unit testing is a technique for unit testing application code that relies on services provided by the run time environment. The application code that uses services provided by the run time environment henceforth will be referred as server side code. In order to test server side code, tests must execute in the real environment (production run time environment). This testing is commonly done as part of the application integration process after the code is deployed. The essential value of executing such tests is to assure that when the code is deployed in the real environment, the server side code will work as expected. Furthermore, this testing is important because no matter how good a test environment is, the server side code is likely to run differently because of factors such as other components in the real environment may interact unpredictably with the server side code. To achieve this assurance, server side code and its interactions with the real environment should be completely tested.

According to Sheldon Wosnick (2002), in-container testing possibly can provide the middle ground between code logic unit testing and functional unit testing, assuring the application will run when deployed. (Cactus) is a framework that implements in-container testing strategy, for conducting integration unit testing of server side code (servlets, EJBs) (Appendix B).

Although, in-container testing is important it adds increased overhead to the testing process. One reason is the overhead of starting the run time environment which takes time depending on the nature and complexity of the environment. Another reason, for increased overhead is because deploying the tests and executing them against the real environment increases the test execution time. Consequently, this testing cannot be used as part of fast compile and regression testing strategy. In addition, TDD's reliance on quick feedback from tests cannot be achieved using the integration testing strategy. A contrasting approach, to in-container testing of server side code is an out-of-container testing strategy. An out-of-container testing strategy uses MO approach which is useful for logic unit testing in the development environment.

2.5.4 Functional Unit Testing

The functional unit testing technique is especially relevant in the domain of Web applications and also known as a black box testing technique. Web applications are composed of a set of pages that together accomplish one or more functional requirements. To test functional requirements of Web applications, pages achieving the functionality are components that need to be tested at the **unit level** and fall in the scope of a unit test as proposed by (Lucca et al., 2002). Test cases are designed on the basis of the **functional requirements** of Web application. This technique is referred as functional unit testing (Pipka, 2002; Massol and Husted, 2003) because it overlaps both the area of functional and unit testing. Figure 2-4 shows the scope of functional unit testing in the context of Web application testing.

A lot of work is done in the area of modelling Web Applications by representing entities of Web application as objects and their structures, relationships and dynamic behaviours (Kung et al., 2000 ; Lucca et al., 2001 ; Lucca, Fasolino et al., 2002 ; Lucca et al., 2004). These models provide the basic strategy for deriving functional test cases automatically. Functional unit tests are executed in several ways such as by manual testing involving human interaction with Web applications through the Web browser verifying the

behaviour of the application. Secondly, by using record and playback (R & P) testing frameworks that record actions of a manual tester and verify expected output of the application. Tests are automated by replaying the actions and comparing actual output with the expected output. In addition, the functional unit testing approach is automated for executing functional tests by frameworks such as (HttpUnit), (HtmlUnit), (jWebUnit) and (Canoo). The black box testing approach implemented by these frameworks provides a means for simulating an end user request and then queries the response returned by the Web server to verify that it is correct. More details on these frameworks are provided in (Appendix B).

The effort to maintain automated functional tests is high because functionality of Web applications changes quickly over time. As a result, functional unit tests are easily broken over time. For example, an automated test input may be a page element that changes over time and the test output is based on analyzing these elements. Consequently, making a change to any page component will break the test and requires effort and time to refactor the associated tests.

2.5.5 Performance, Stress and Security Testing

Performance testing and load testing are used to assess Web application for:

- 1) Handling expected loads caused by concurrent users
- 2) Acceptable response time

Performance testing uses load testing techniques for measuring and benchmarking Web applications under various load levels. This helps in detecting bottlenecks within the Web application components. Load testing is a part of the performance testing process and is defined as the process of exercising the system under test by providing large tasks as input (Loveland et al., 2004). Commonly used lightweight tools that automate Web application performance and load testing are (JUnitPerf), (JMeter), (Mercury and LoadRunner).

Stress testing of Web application is defined (Loveland, Miller et al., 2004) as the process of subjecting a system to an unreasonable load taking away resources (for example RAM) needed to process that load. The goal of this testing technique is first to stress the application enough to find conditions that will break the application. Second, it determines the failure behaviour of an application without adequate resources in a suitable manner (for example not corrupting or losing data). Testing techniques used for stress testing of Web application are a combination of load testing tools together with ways of applying stress to the application; for example by running processes that consume high resources (CPU, memory, disk, and network) on the Web and database servers.

Web Security testing (Nguyen, Johnson et al., 2003) determines vulnerabilities and information leaks caused primarily due to incorrect programming practices, mis-configuration of Web servers and application specific servers. Web security testing strategies are:

- Testing access control of Web application to determine whether the single class/classes of users have correct access privileges to the application
- Testing how secured the Web application is to handle client data (data integrity)
- Determining whether the user identity verification is implemented correctly.

2.5.6 Web Application - Model, View and Controller Testing

This section describes the current state-of-the-testing in Web application development with respect to MVC model because this model is apt to support testing during all development stages (Pipka, 2002). In addition, an MVC layered approach for testing provides support for TDD of Web application code.

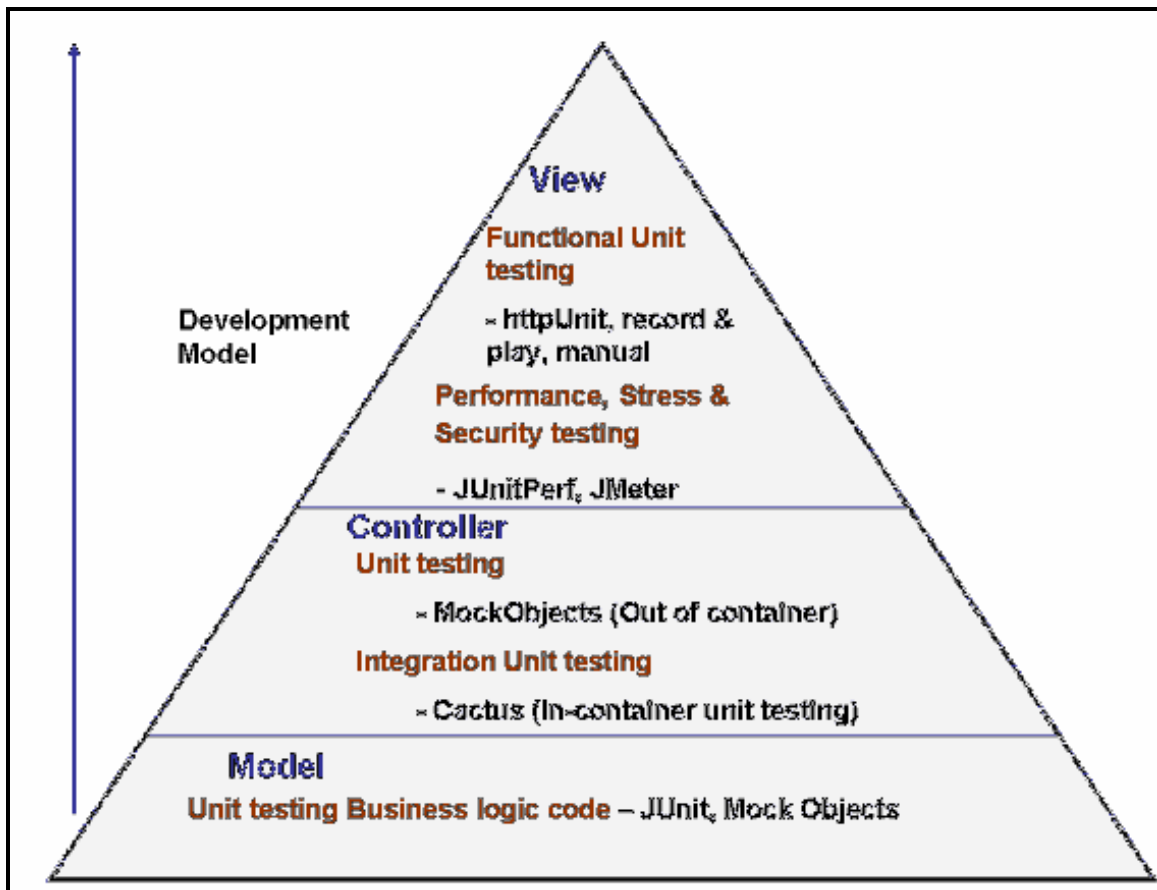


Figure 2-5: An overview of testing techniques for Model, Controller and View layers.

Figure 2-5 summarises testing techniques and commonly used tools to automate testing using MVC model of development. Accordingly, the test activities are divided into three scenarios model, controller and view testing based on the order of implementation of these layers.

Methods implemented in the model layer contain logic to manipulate the application data. These methods are tested using unit testing techniques ensuring that each unit of software works correctly independent of other units. Functionality of this layer is verified by comparing the expected predetermined output with the actual result. JUnit automates unit testing of the model layer implemented using Java. Moreover, testing can be closely integrated with the development process. Additionally, it is possible to build test suites

incrementally and provide developers with a regression testing strategy. The MO approach can also be used for unit testing the model layer.

Components of the controller integrate both model and view layer by communicating with both. Components of this layer run within a container that provides services such as security, life cycle management of the component code, logging and persistence. Servlets in J2EE are components that form the controller layer. With respect to testing this layer, there are two techniques commonly used. The first technique is unit testing using the MO approach implemented by tools such as (ServletUnit), which allows a TDD development process. Secondly, integration unit test of servlets is commonly done as part of the application integration process using (Cactus).

The final testing layer is the view responsible for presenting the application data through objects created by the controller components to the end user. View layer components are represented as JSP and HTML pages. As part of testing, the functionality of these pages must be verified. Additionally, the client-side functionality that runs inside the browser, for example JavaScript code must be tested. HttpUnit supports client-side testing. In order to test the view layer, testing techniques simulate client requests and verify the server response. Functional unit testing approaches explained in (section 2.5.4) apply for testing this layer.

2.6 Summary

In this chapter, I have briefly summarised portal application technology and its key components. Portal applications can be viewed as a special type of Web application. Considerable differences stem from distinct, complex running settings, services provided by the run time environment and new components called portlets that form the building blocks of portal applications. These differences motivate the need for novel testing techniques. Previous research has focused mainly on Web application testing process. Investigating, existing testing techniques provided insight on, whether it is possible to adapt

existing testing techniques from Web application to portal application domain. Portal application testing process is an area that still needs to be explored although existing techniques from Web application testing can be adapted and reused. Chapter 3 presents the results of case study conducted to understand the state-of-the-practice surrounding testing process for portal applications in the industry.

Chapter 3. Portal Application Testing: Case Study

In this chapter, I discuss the case study conducted to evaluate aspects of testing portal applications, namely testing techniques, methodologies and automated testing tools used in the context of a company. First, I outline the overall objectives of the study and the research questions the study addresses. Next, the context of the study, participants and data collection strategies used are described. Then, the results of study are discussed. The chapter concludes with a brief summary of the study findings.

3.1 Objectives

A prerequisite to providing support for better tested applications is an early assessment of existing testing process. The inspiration for the empirical study conducted came from this need. The study objective was describing existing testing practices and not hypothesis testing or validation.

Objectives of the case study are:

1. To **explore** how portal applications are tested by presenting the main techniques and methods currently used.
2. To **evaluate** the testing practices in use and identify challenges in testing portal applications.

Specific research questions that guided this study are:

1. How do portal developers test different aspects of portal applications (techniques and tools in use) (*objective 1*)?

2. What is the nature of challenges that hinder comprehensive testing of web portal applications (*objective 2*)?

3.2 Case Study Methodology Overview

The research study conducted is classified as interpretive (Walsham, 1995) since it embodies the philosophy that the knowledge of reality can be gained through social constructions such as language, shared meanings, documents, tools, and other artefacts. Interpretive research does not predefine dependent and independent variables, but focuses on the complexity of human behaviour as the situation emerges (Kaplan, 1994). This was suitable to explore how portal applications were tested by the developers. Case study research is defined as an '*empirical inquiry that investigates a contemporary phenomenon within its real-life context*' (Yin., 2003). A case study was appropriate because it helped in exploring contexts to gain a better understanding of how developers tested and engineered portal artefacts in the company. The units of analysis for the case study were the testing, development and deployment processes practiced by software development teams for portal applications.

3.3 Study Context and Participants

The case study was conducted in collaboration with Sandbox Systems to explore how developers tested and engineered portal applications. The company had a team of five software developers with three to five years experience working as consultants. A period of approximately three months (November 2003 to January 2004) was taken to understand the current testing process and the challenges in building and testing portal applications.

Sandbox developers were developing and maintaining the company's internal portal Website as well as "external" enterprise portal applications developed for other companies. These applications focused especially on developing portlets for enterprise portals. Portlets were designed, developed and tested using the IBM-provided Websphere portlet toolkit test

environment within the integrated development environment called Rational Application Developer. Applications were deployed for production use in WPS portal server supported by IBM. Sandbox consultants were integrating functionality provided by existing legacy applications into the portal framework, using services provided by the portal server especially single sign on. Typically, this functionality was integrated by developing the interface methods (Model layer) for the existing code. Portlets were used to invoke these methods.

3.4 Data Collection and Analysis

The data collection methods included interviews, notes taken and numerous discussions with the chief architect responsible for developing portlet based e-business tools. This was permitted by the research agreement between Sandbox Systems and the University of Calgary. The nature of the interview was unstructured without a formal protocol although a basic guideline for questioning was devised by me. This allowed the study participants to steer the interview while describing testing and related issues. These discussions were recorded, and provided deeper insight into the development and testing practices. Data for interpretation comes mainly from the transcript analysis (Appendix G) of these discussions, feedback reported, recorded interviews and notes gathered. Furthermore, to gain understanding about the nature of tests written, and run by developers an existing portal application built by the company was inspected.

3.5 Results

3.5.1 Testing Practices in the Company

Testing practices in the company are illustrated in Figure 3-1. Methods in the Model layer were unit tested using JUnit. In some cases, Mock objects were being used to represent database dependencies for unit testing. The expected output from the unit tests validated that the Model layer methods were giving the correct output. In addition, developers

reported that TDD was used to develop the Model layer wherever possible. A single developer implemented the application incrementally because applications were small in size. The unit tests were executed frequently inside the WSAD integrated development environment (IDE) to ensure, that adding new functionality has not broken existing code. On the other hand, portlets were being tested manually by deploying the application in the portlet toolkit test environment installed in WSAD IDE (Figure 3-1).

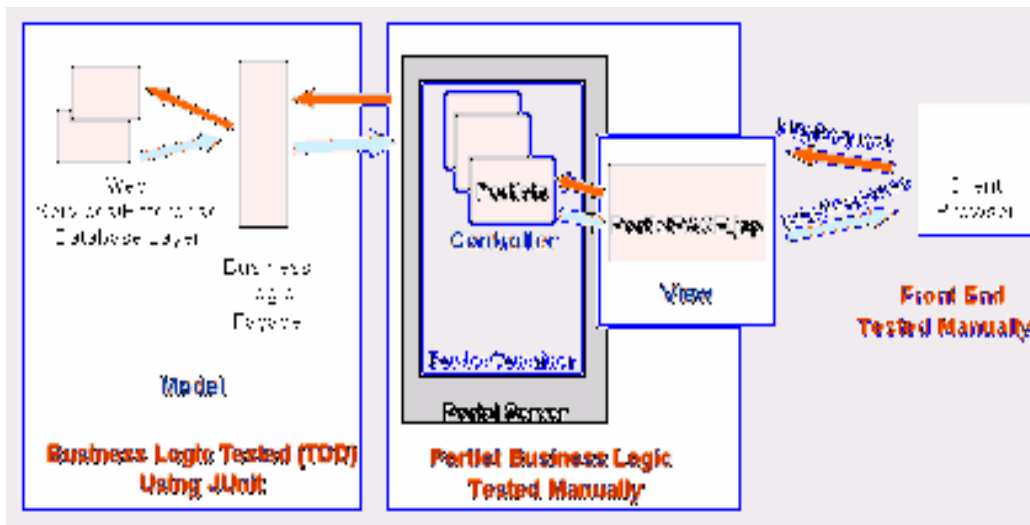


Figure 3-1: State-of-the-testing practices at the company according to model, view and controller layers.

Integration and functional testing of the portlets was done manually in the staging portal server prior to deploying the application in the production portal server. No automated functional testing techniques were being employed to write or execute the tests. The reason reported for manual testing of portlets was that no tool support for writing and executing tests was available. Consequently, portlets were tested for functionality manually in the production environment by simulating a user role, and making sure the portlet rendered the appropriate content. The developers also reported that during migration of the portal application to the production portal server, portlets did not deliver functionality i.e. they sometimes rendered as empty windows.

Performance and Scalability tests for portal applications were not performed at the company. User acceptance testing was done by setting up test user accounts based on the business unit that requested the functionality. These test users tested the functionality by navigating the portlets and links rendered within the portal pages to verify that the application satisfied all the requirements from user perspective.

A portlet application built by the Sandbox developers was inspected for the number of unit test case methods versus the number of existing methods. This application provided employee training information using employee number as input; by retrieving the information from the Domino Lotus database layer. It was developed using the portlet struts framework (Struts) and followed the standard layered architecture using MVC. Application characteristics for this portlet application in terms of the number of classes, methods, lines of code are presented in Table 3-1. However, it is important to note that the size of a single portlet application deployed in the portal server is relatively small. Many such small portlet applications are deployed within a portal server. These applications are aggregated and presented as the final response portal page to the end user.

Table 3-1: Portlet Application Characteristics.

No of Packages	5
No of Classes	8
No of Methods	39
Lines of Code (LOC)	561

Unit tests in this application first verified successful connection to the database. Next, these tests verified whether the data could be read and written after checking that the database connection was established. Figure 3-2 shows the number of methods in each of the application layers versus the number of test methods. The number of test methods (Figure 3-2) was counted by me because the application was not large. Test coverage tools such as (Clover) could not be used due to domain dependencies in the tests on Domino components. Figure 3-2 shows that 40% of the Model layer methods were tested. The portlet method (which in this case was a single “*execute*” method of the struts based portlet

class) was not being unit tested because of a lack of automated testing framework that supported writing and executing portlet test cases.

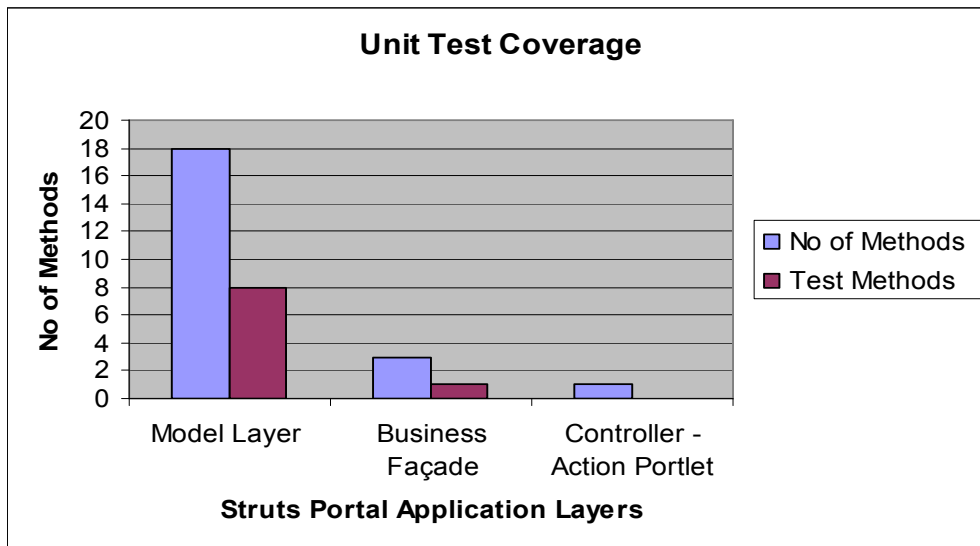


Figure 3-2 Case study- number of methods versus test methods in the portlet application.

3.5.2 Challenges in Testing Portal Applications

One of the challenges highlighted by portal developers was a lack of a direct way to conduct fine grained unit testing of portlets using existing testing frameworks. Although, setting up an automated unit test for a method that retrieved data was possible, it was difficult to write an automated test to prove, that the data would be presented by the portlets. In addition verifying whether the data was correct was difficult. One of the developers highlighted the need for a portlet testing technique that could provide “*a way to be able to put in hooks into the portlet stream to know whether it is valid and that your expected output is what you need*”. Another, specific issue highlighted was that portlets did not render data which was referred by the developer as, “*empty portlet window*” problem. This reflected problems with the data packaged by the portlets running in the context of the

portal server because the portal handled rendering of the data fragment as the view. The developer reported that,

“Something valid is getting deployed is not known, until you bring down the environment, redeploy, fill the debug on at the portlet level to understand the reason portlet fails to render”. --Sandbox developer

Diagnosing and fixing the “*empty portlet window*” problem was reported as a very tedious and time consuming because it required skimming through log files and exception handling provided by the portal server. The developer described the gravity of the problem especially at the time of integrating the application in the production portal server by stating that,

“In the test environment it (portlets) worked. Deployment of these portlets on the production box did not work. There was nothing in the logs and we had no means of knowing what the problem was”. --Sandbox developer

Another challenge highlighted was an absence of an automated framework to simulate various user roles to test the context of portlet behaviour, and to verify access permissions assigned to portlets. It was reported that the administrator assigning the permissions logged in as a user related to each role to verify whether the permissions were assigned correctly. Switching roles quickly was a challenge.

Developers at the company also reported that they preferred to use TDD for developing and unit testing portlet code. In the developer’s words “*developing portlet code in a TDD way would be nice*”. Furthermore, it was reported that the overhead of testing changes in the portal server environment was high. One reason was that the portal server deployment test environment took a long time to initialize. As a result, deploying the application code was time consuming. Therefore, the development, deployment and test cycle of portlet applications was longer than acceptable for TDD. In the developer’s opinion, initializing the portal server environment each time, took approximately ten minutes of time, which aggregated to a lot of time and loss of productivity when done multiple times in a single day.

“Bringing the server down becomes prohibitive and it is a painful process; costs a unit of time approximately ten minutes which is a lot of time on the project”. --Sandbox developer

3.6 Case Study Limitation

The results of the case study provide an understanding of the practices employed by portal developers at the company for testing portal applications. However, the case study assessment suffers from some limitations. In retrospect, a case study where one of the researchers was allowed to spend time in the company would promote deeper insight into the company testing processes and techniques in use for developing, deploying and testing portal applications. The case study results reported are based on several interactions with developers working for the company and notes recorded over this time, and were subjective opinions of the developers. Reinforcing the results with work place observations would improve the study. Another, limitation is that there was little quantitative data gathered in the study. The ability to track test code coverage metrics, developer activity and time logs to calculate time spent on testing, debugging and deploying portal applications would be useful. While gathering this data would provide a more objective measure, the collection process for these metrics would impact the natural flow of developers work and was not acceptable for the company.

3.7 Summary

In this chapter, I first presented objectives of the case study conducted to explore testing practices in the company. Next, I detailed the study context and participants and present results of the study. Results highlighted challenges in automated unit and integration level testing and present a set of implications for developing testing practices specific to portal applications. In the next chapter, I will discuss results of the survey conducted with portal developers and present answers to the specific research questions raised in Section 3.1.

Chapter 4. Portal Application Testing: Survey

In this chapter¹, I discuss the design of a survey conducted to provide broader insight into testing practices for portal applications. The survey was conducted to strengthen the case study result using an empirical assessment technique called methodological triangulation (Patton, 2002) which combines multiple methods of qualitative inquiry. The triangulation technique is based on the premise that each method of study reveals a different aspect of empirical reality.

In this chapter, I first present demographics of participants and their background. Next, the results are provided and interpreted by analysing the survey responses. The two research questions outlined in Chapter 3 are answered in results of the survey. I conclude this chapter with a discussion of the limitations of the survey methodology.

4.1 Survey Methodology Overview

To extend insights into the portal testing process, a survey was conducted in the context of interpretive research (Lee, 1997). Fetterman (1989) describes “*survey questions, in*

¹ A part of this chapter is published as:

Bajwa, H; Xiong, W. and Maurer, F. (2005) **Evaluating Current Testing Processes of Web-Portal Applications**, Proceedings of International Conference of Web Engineering (ICWE 2005) LNCS, Volume 3579, Jul 2005, Pages 603 – 605

interpretive research, can lead to survey responses that constitute the material the researcher uses to help develop a thick description and rich understanding of the life world of the survey respondent". The larger research perspective was designed to understand how portal developers tested and engineered portal applications in different companies using portal technology. In addition, the survey method of inquiry provided the required breadth for understanding the testing practices which the contextual case study lacks. Allen Lee (1997) also reported that "*surveys are good for complementing other sources of data like documents, observations, conversations and also help in providing materials for interpretation for thick description and for developing theory*". To this end, the survey was suitable in augmenting the knowledge gathered from the case study discussion and interviews on the testing process of portal applications. The overall survey objective was to answer the question how developers in the industry currently test portal applications. An added motivation for the survey was to validate the perceived need for an automated tool support as reported by the company study (refer section 3.5.2). I wanted to identify whether, "*other*" portal developers experienced challenges in testing and deployment of similar dimension as the company case study. In short, the survey was an attempt to validate the scale of problems and make the results of this assessment stronger by generalizing to a wider portal community.

4.2 Survey Design and Sample Selection

The survey was designed to answer more specific questions using the case study findings to guide the design of a few survey questions. The questionnaire inquired about the specific techniques and testing practices in place, challenges in testing and the deployment process of portal applications. As outlined in (Appendix C.2), the questionnaire items solicited input on how testing was done and whether the testing employed was automated or manual. The survey also solicited comments to some open ended questions where the developers were asked to describe particular challenges in testing portal applications. In addition, an introductory section was included to gather background of the

developers, their companies as well as the nature of portal applications developed because it impacts the extent of testing needed. The background section also captured information of the companies with respect to development and testing teams to inquire whether a separate quality control and testing teams were in place. The survey activities were designed by closely following guidelines as suggested by (Pfleeger and Kitchenham, 2001).

I “tested” the initial survey on two known portal developers and based on their response the questionnaire design was refined. The survey sample selection process used a non-probabilistic sampling technique called “*convenience sampling*” (Kitchenham and Pfleeger, 2002) where the participants were selected because they were easy to access and had a good chance of representing the population. The participants chosen were from the pool of portal developers working in the industry by sending e-mail requests on portal discussion forums for JSR 168 portlets and portal servers and communities of practice such as portlet yahoo groups and java based (PortletCommunity). Some portal developers invited to participate in the study were amongst the community leaders in portal technology.

The survey was administered via e-mail together with a letter providing a brief overview of the goals of research (Appendix C.1). At the end of the survey, the participants were asked to indicate their choice for participating in a telephone interview. Responses collected were received over three months from January 2005 - March 2005. Following this, I conducted telephone interviews that provided an opportunity to explore answers to more open ended questions. The nature of questions asked for example, was how a typical portal application is implemented (design, testing, development and deployment) in the developer’s organisation. Typically, the questions asked in the telephone interview were guided by the survey responses to clarify some answers the respondents had provided to the questionnaire. Prior to conducting the telephone interview, I had conducted an analysis of each of the responses, and outlined a set of issues to be explored with each respondent. This outline served as a checklist for the telephone interview.

4.3 Response Rate

The questionnaire (Appendix C.2) was e-mailed to 150 portal developers. Responses were received from a total of 20 developers. Five out of 20 responses were discarded because they were inadequately completed. Therefore, 15 valid responses are being used to present the results. Responses received were subjective in nature and were analyzed and interpreted by me. Telephone interviews were scheduled and conducted with 7 out of 15 respondents.

4.4 Participant Demographics

Data was gathered on participant experience in developing both Web applications as well as portal applications. It was important to understand the experience in both technologies because portal technology is an extension of existing Web application technology. As a result, experience and knowledge of Web application development, test and deployment process can be applied to portal application domain.

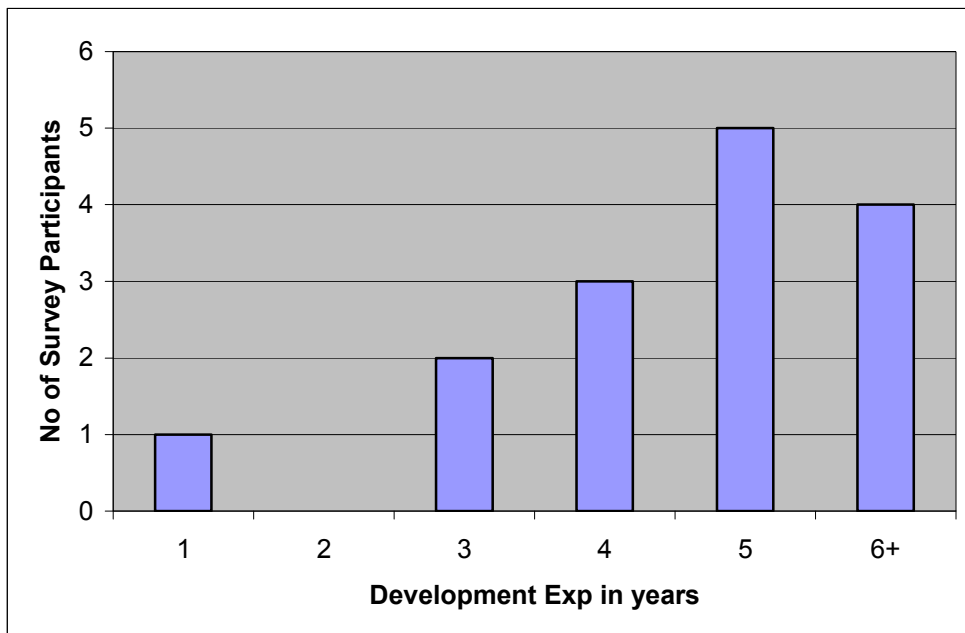


Figure 4-1: Web Application Development Experience of Survey Participants.

Of the 15 survey respondents, 9 (60%) had between five to six and more years of Web Application experience (Figure 4-1). Of these 9 respondents (60%), 6 had between three to four years experience whereas the other 3 had two years experience working with portal technology (Figure 4-2). Three respondents (20%) had four years of Web application technology experience and between one to two years of using portal technology. Only 3 respondents (20%) had between one to three years working with Web application technology and one year of portal technology.

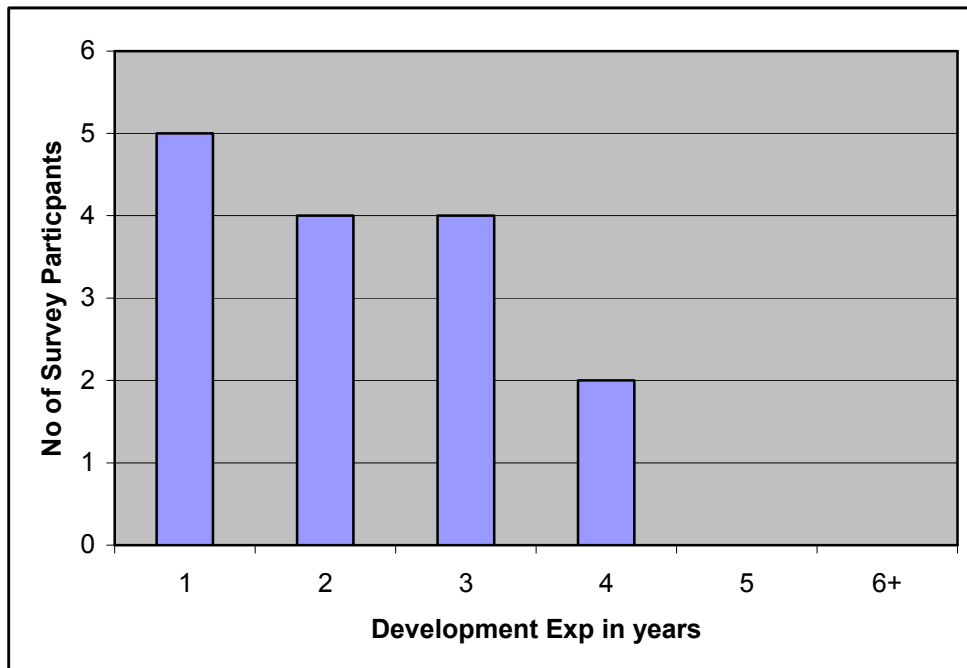


Figure 4-2: Portal Technology Experience of Survey Participants.

Most of these participants in the survey were working for a range of positions in their companies such as J2EE portal architects, portal application developers, portal server implementers (responsible for architecting open source and vendor specific portal servers), and technical assurance managers. One survey respondent had participated in the portlet JSR standardization effort and another one was a lead in implementing portal server for Sun and also a java.net community leader.

Testing is an activity that closely ties in with the development process. Therefore, it was important to find out the nature of development process used by participants. Eight respondents reported using extreme programming (XP) (Beck, 2000) and its modified versions for portal application development, whereas 5 developers reported using the rational unified process (Krutchen, 1999) which is an incremental and iterative development process. Another participant reported that they were using an internal development process called Global Services development methodology developed by IBM (UserEngineering) which uses an incremental approach to delivering business solutions. This process focuses on user centered design, early user involvement and end user testing. Only a single participant reported that their company had no defined development process.

The type of portal applications developed by the participants is classified into four categories based on the nature of services provided by portal applications to the end user.

1. ***Portlets for Integrating Enterprise Applications***: Eight participants reported that they developed enterprise portal applications to integrate access to organisation specific information, for example documents, real time data feeds, business processes, collaborative support for their company's applications. These applications were developed by providing the portlet front end and accessed by using the single sign on feature of the portal server.
2. ***Custom Portlets for Portal Server Vendors***: Three survey participants reported that they implemented portlets to provide custom functionality for the portal server. These portlets were built-in with the portal server, for example collaboration and e-mail portlets.
3. ***Administrative Portlets***: Another participant was working on portlet development for IBM's internal portal website. The portlets developed were providing administrative functionality to create, delete portal pages and to set up user constraints on the type of portal pages created.

4. **Portlets for Academia:** Another participant reported developing portal applications focused on offering various services to the university community, for example announcements, web mail, news, bookmarks, classifieds, and class lists. Many of the portal applications were interfaces to existing Web applications, whereas 2 other participants reported that they developed portlets for grid enabled portal application. These portlets allowed high performance resources to be used by a single sign on access to portal server.

4.5 Portal Server Deployment Infrastructure

The type of portal server used for application development, testing and deployment and the underlying vendor specific portal technology is important for automated testing. Participant responses indicated a range of vendor specific portal servers used (Table 4-1).

Table 4-1: Type of Portal Server Environment Used.

Development Environment (developers implement & test portlets here)	Production Run time Environment (portal applications are deployed here)	No of Survey Respondents
Websphere Studio Developer Environment (5.1) Portlet Toolkit	IBM Websphere Portal Server 5.1	7
eXo Platform 1.0	eXo platform 1.0	1
BEA Portal Workshop	BEA Weblogic Portal	2
Gridsphere	Gridsphere	2
Uportal	Uportal	1
SAP NetWeaver Portal	SAP NetWeaver Portal	1
Portlet Builder with Sun	Sun One Portal Server	1

4.6 Results

Question 1: How do portal developers test different aspects of portal applications (techniques and tools in use)?

The results of the testing techniques reported across the survey are summarised in Figure 4-3.

4.6.1 Unit Testing

Thirteen respondents (87%) reported that unit testing of portlet service layer methods was performed using JUnit (Figure 4-3). Portlet service methods are those methods that provide a specific service to portlets, and are invoked by the portlets. Two respondents (13%) indicated that no automated unit testing of these methods was conducted.

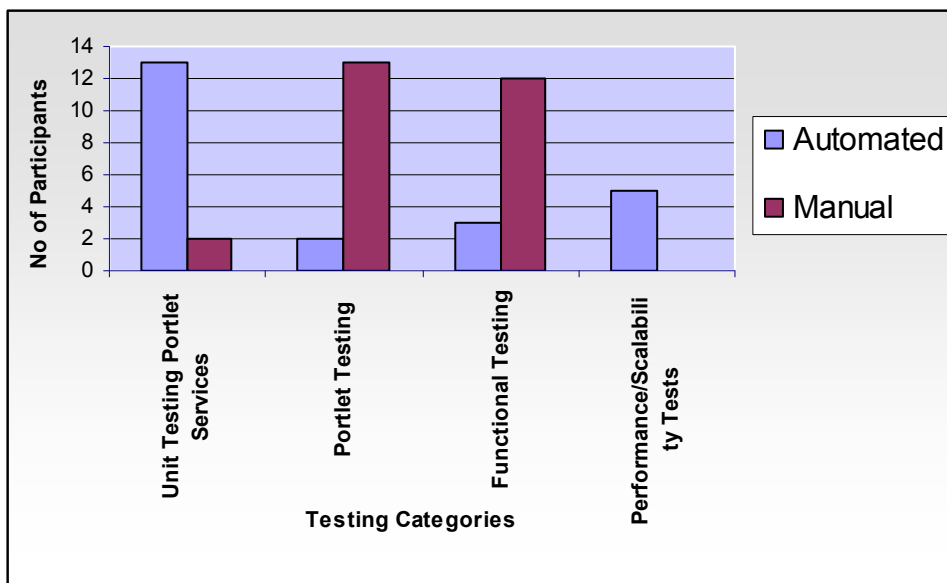


Figure 4-3: Survey results showing automated versus manual Testing.

A single respondent described that their company used the mock object unit testing technique for mocking domain dependencies. On the other hand, the other respondents did not indicate any information on how unit tests accessing backend dependencies were written and executed. In addition, it was reported that test cases were written and executed frequently during development by the developers, and sometimes unit tests were executed as acceptance tests by business analysts and end users.

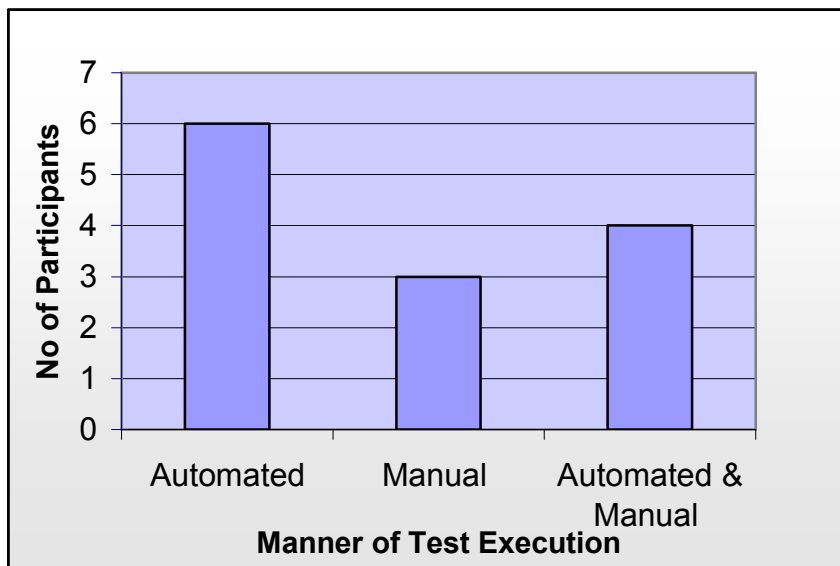


Figure 4-4: Survey results showing how unit test cases are executed.

The manner in which Unit tests were executed is classified as either “*automated*” or “*manual*”. An automated test execution is a part of the continuous integration process where the developers integrate, build and test the system many times a day, every time a task is finished for continual regression testing (Fowler and Foemmel, 2005). Continual regression testing means that no existing functionality has regressed as a result of the changes in the code. Since the continuous integration process executes many times a day, it is automated using tools such as (CruiseControl). The key benefit of the continuous integration process is that teams get continuous, early feedback and integration testing is parallel to development (Fowler and Foemmel, 2005). In contrast, a “*manual*” process means that the test execution is not integrated as part of the continuous integration process. Six respondents out of 13 (46%) (Figure 4-4) reported that they automated the unit test execution process used tools such as (CruiseControl), (Maven) and (ANT). Three respondents reported that unit test execution was done using the JUnit test runner (manual) because the effort and the resulting benefits to set up an automated continuous build processes was considered feasible only for larger applications and for larger teams. Four respondents reported that tests were executed both using the JUnit test runner and as part of

the continuous build process many times a day. Many respondents thought that comprehensive unit testing of the backend business was the key to minimizing errors in the portlet layer of the application.

Thirteen respondents (87%) reported (Figure 4-3) that portlet components were tested manually (no portlet unit tests were written and run). Only 2 (13%) respondents conducted automated unit testing of portlets. One of these 2 respondents indicated that the portlet testing framework used by their company was a part of the portlet container and was based on adapting (HttpUnit). Another respondent indicated that they used (PortletUnit) that also extends HttpUnit and the (Pluto) portlet container for unit testing of portlets.

One reason reported for testing portlets in a manual manner was a lack of tools that support executing portlet unit tests in the portlet container. In one respondent's opinion,

“There are no tools available for portlet testing in container, so we do not use any server side testing frameworks. But each portlet is tested through WSAD Portal toolkit for functionality”. -- Survey Respondent

Another reason indicated for testing portlets manually by one respondent was that, *“there is no easy way to test doView in an automated way because it requires container parameters”*.

A respondent indicated that portal applications in their company were designed so that methods in the portlet class invoked either methods from the other backend classes or private methods defined within the portlet class. The methods in the backend classes and the private methods included the complex logic that required comprehensive testing and the portlet API methods were limited to invoking them. In other words, very minimal logic was included in the portlet API methods to merit testing. The methods in the backend classes and the private methods were tested using standard unit tested techniques because these methods did not access container provided services. In the respondent's opinion the portlet API methods should not require any unit testing and using comprehensive unit testing prior to calling the portlet methods should be sufficient,

“The doView method for all intent and purposes is for running JSP's, you already have done JUnit testing ahead of calling those particular methods”. -- Survey Respondent

4.6.2 Functional Testing

Three respondents (20%) reported (Figure 4-4) that they performed functional testing of portlets by automating the process of writing and executing functional tests. These respondents were using httpUnit adapted to specific portal server technology used in their respective companies. The remaining 12 respondents (80%) reported that this testing was done cursorily wherein a developer or an end user tests each functional area of the portal application by conducting a walk-through of the functional requirements. Two respondents informed us that they were testing functional requirements indirectly by unit testing using junit. In their opinion this manner of testing was fulfilling integration functional testing in a limited way; according to the respondent, *“we do functional testing in a limited way; start to end tests using junit”*. In one respondent's opinion, an effective automated functional testing process was dependent on the complexity of the backend integration process. As a result, the complexity of integration determined the balance between manual and automated functional testing, as reported that,

“Depending on the complexity of integration, you should be able to test to an extent but you may not get all the way through because you are relying on SAP, DB, and Oracle in the same application to return information from all these environments”. -- Survey respondent

4.6.3 Performance and Load Testing

Five respondents (33%) reported that performance and load testing of portal applications was being conducted (Figure 4-4) in their respective companies. HttpUnit which is a functional testing tool was extended and used in the case of three respondents to simulate multiple clients for this testing. Two other respondents reported that they were using Load

Runner and Mercury interactive tools for load and performance testing. Ten respondents (70%) had no response for this question. One specific respondent indicated that, “*automated test support would be very useful in performance and load testing of portlets.*” The results also revealed that only those respondents who conducted automated functional testing techniques also performed load and performance testing using the functional testing tools already in place.

4.6.4 Portal Application Deployment

In response to the question that inquired whether deployment related errors were common when the portal application was migrated from the development environment to the production portal server 11 out of 15 respondents reported errors of different severity levels. I inquired specifically in the telephone interviews to understand the nature and severity of errors. Four respondents indicated few and less severe errors. Also in this case, portal applications were developed and tested using identical production and development environments. Therefore, as reported few errors were experienced during the application integration phase because the underlying infrastructure was the same in both development and production. It follows from this result that having an identical test and staging environment is important. During the interview, I also explored the process of portal application deployment. The process described was automated using a build script to deploy the war files to the production environment. In addition, steps for setting up the backend infrastructure were automated. The administrator monitored the build and eliminated components that caused the build script to fail. Moreover, manual testing was employed to ensure that the portlets were functional.

The scale and severity of the errors at deployment time reflected by many responses was difficult to categorise in a comprehensive manner. However, the errors are classified based on survey responses. Typically, the portal applications had errors at deployment time (when the application is being integrated) or after deployment at run time when the application is invoked for functional testing.

- Deployment time: Six responses suggested that errors during application integration occurred due to incorrect deployment environment configurations (unresolved external component and resource references) and application configuration parameters, for example missing deployment descriptor entries. According to one respondent,

“Errors were infinite in possibility. The team is invariably involved as it is most often a system configuration issue that causes the problems such as differences in the Engine settings”. –Survey Respondent

- Run time: Ten respondents reported that errors during application integration occurred due to missing class files which were needed at run time by the portlets. Consequently, the portlets failed to deliver functionality and exceptions were recorded in the portlet logs. According to one respondent,

“Jar conflicts are probably the most severe. Sometimes we will have to shuffle jars around between the Web application and shared libraries to get things to work. The jars in question are mostly related to APIs that we use, usually security and we have to pass security credentials between portlets so this is a necessary thing.” –Survey Respondent

4.6.5 Challenges in Testing Portal Application

Question2: What is the nature of challenges that hinder comprehensive testing of web portal applications (goal 2)?

In response to the question that asked the developers to describe the challenges in testing portal applications most of the responses pointed to the need for a tool that automated unit testing of portlet methods. The key challenge was inability to test the “*portalish*” behaviour of the portlets which represents the portlet code that relies on services provided by the portal server.

The responses are classified on the two most often indicated testing techniques for unit testing of portlets namely the testing approach using portlet mock objects and portlet testing using the portal container and portal server context.

Table 4-2: Survey responses indicating the testing techniques needed.

	Portlet Unit Testing - Out of Container(n=13)	Functional Portlet Application Testing- Incontainer tests (n=12)
Yes	7 (54%)	8 (60%)
No	2 (15%)	3 (25%)
Not known	4 (31%)	1 (8%)

The total number of respondents in each category depicted in the Table 4-2 does not include respondents who earlier indicated that the automated testing process for portlets and functional testing were already in place. These respondents did not indicate any response to this question. 8 respondents stated that having an ability to run portlet tests would be valuable in performing portlet functionality checks.

The other 3 respondents (25%) thought that running tests inside the portal server production environment was not desirable because production data was likely to change. Another respondent indicated that automated approaches were not valuable for out of box component technologies. 7 respondents (54%) asserted that portlet development using an Out-of-container approach with mock objects would be useful. One reason for mock portlet approach was the problem in the company environment indicated by the response that,

“Developer productivity, in our portal environment, even minor changes take a long time to test due to having to deploy and wait for server cycles, etc. Using a mock object tool would allow to develop and test a portlet using a much faster compile and run strategy. By removing the complexity and overhead of repeated deploying, developers can be much more productive.” --Survey Respondent

An automated tool to support portlet testing was important and reasons indicated by the respondent was that,

“Server side testing is really important because we never get a chance to truly test our portlet code; just the supporting code that gets used by the portlets is tested”. -- Survey Respondent

One of the reasons reported, for portlet testing tool support stated that an automated way of testing would likely reduce the time and effort for testing and debugging portlets. As a result, more time could be focused on design and implementation activities indicated by the comment that, *“the more we can automate testing (especially testing of portlets which doesn't seem to exist at the moment) the more time we can save and use for actual design and implementation, would like to have a way to test portlets using either in-container or some sort of portlet mock object suite.”* -- Survey Respondent

Another reason which supported the need for automating portlet testing reported was that debugging portlets meant sifting through error data logs to diagnose the reason the portlet code failed

“Portal technology we use has an utility that gives logs of data that is dumped while running. This is like skimming log files and usually only contains generic information”. – Survey Respondent

Another respondent stated that the need for writing non-exhaustive portlet tests was important in a way that,

“Smoke test that a portlet is going to show up that you will not get a portlet exception that causes things to not show up at all and return valuable information when testing success or failure of a response”. – Survey Respondent

The need for a framework was augmented by another response,

“Having a tool that verifies the portlet functionality -- is definitely a worthwhile tool for developers and deployers”. – Survey Respondent

4.6.6 Interpretation

This section provides important factors revealed by the survey responses and results of the case study. These factors provide potential answers to the question why there are errors at deployment time and reasons for long application deployment cycles and high manual test effort for portal applications.

4.6.7 Design and Testability of Portlet applications

The need for performing comprehensive unit testing of the portlet layer is dependent on the amount of complex business logic in the portlet layer. Therefore, as far as possible in case of portal applications, the business logic code should not be tightly integrated with the portlet layer. Another important aspect of portlet design is that all the logic related to the Model layer should be processed and encapsulated in a separate object (java bean for J2EE portlets). The portlet must be designed to access this object. Although, this is a good design practice, many times it is difficult to separate the business logic associated with services provided by the portal server, for example portlet business logic specific to user preferences. In some cases, portlet methods contain a lot of logic associated with event actions on portlet windows such as minimize, maximise, configuring different portlet modes which must be tested. Therefore, the extent to which the business logic can be separated from the portlet layer is dependent on the portal application specific functionality.

4.6.8 Deployment Process and Environment Complexity

The errors at deployment time in the portal server are largely dependent on many factors. First, it depends on the deployment process and practices in place using proper guidelines for the migration of an application to the production server. Second, the test server used for deployment may not be stable or configured properly. In the case of survey respondents that did not report serious errors related to deployment, it was noted that the development, staging and deployment portal server environments were very closely mirrored. However,

the degree of similarity of different environments is very specific to a company, its available resources and the quality processes in place.

Another, reason for errors reported at deployment time by the study as well as survey (“*empty portlet window*” in the production portal server) were more emphasised in certain product and technology specific portal servers. Severity of errors was related to the complexity of various portal server environments. More complex environments are those that provide more comprehensive portal services. Therefore, portlets deployed in these environments are more prone to errors at deployment time. In other words, complex portal server environments have higher portlet sensitivity and therefore a greater need for testing of portlets.

The results also highlighted a need for a lightweight test portal server which was less complex and supported hot deploy because initializing and bringing down the test environment was time consuming. Requirements for an evolved deployment portal server environment were also revealed as a result of the study. I have reported these requirements because they are an important aspect of improving quality of portal applications and reducing errors at deployment time.

- The **need for a more evolved deployment environment** that supports better exception handling and logging mechanism.
- The **need for a lightweight test environment** that reduces the test, develop and deploy cycle on developer machines. However, the decision to use this environment is highly specific to the company. The lightweight environment will be different from production deployment environments. Therefore, using this environment makes a trade off between shorter development cycles versus longer deployment cycles during the application integration phase. As a result, allocating sufficient time for portal application integration testing is important.
- The **need for improved deployment procedures and best practices** is important because certain types of deployment related errors can only be corrected by following a

proper checklist prior to deployment. Another way to achieve this is to use tools and technologies that support an automated deployment process and track the lifecycle of portlets. For example, Wiley is a tool that helps in looking at the portlet life cycle in the container and diagnoses problems with portlet lifecycle.

4.7 Limitations of the Analysis and Study Methodology

The survey with portal developers added a richer context to the state of the testing practices in the industry and also validated the scale of the problems reported by case study. However, it may suffer from a few limitations. First, the survey responses were received from 15 of the total 150 developers which maybe a low response rate. However, it can be argued that the quality of responses is good because they were received from the population of experts and leaders in the portal development community as opposed to a student population. Second, the qualitative style of the survey with portal developers brought out a variety of subjective responses. As a result, for some replies it was hard to interpret the meaning of the response expressed by the participant although the telephone interview resolved this. Third, survey data was analysed and interpreted by me, therefore may suffer from potential bias of my thinking (tunnelling effect). The nature of the survey and interview questions probed specifically to validate the case study findings. Therefore, they may impact how the responses were provided.

Rough Set theory (Pawlak, Z, 1992), a technique for data analysis was used on the survey data to classify the co-relation between the survey data attributes. This analysis was conducted using the Rose software. The dependent variable selected for the analysis was the attribute deployment error. The objective was to justify based on this analysis the need for a tool that detected deployment errors (in-container testing tool) versus any other influencing factor. The set of attributes and values considered for this analysis is listed in appendix [].

The result of executing the Rose tool on the survey data file generated 11 rules, using 15 data points in the survey data. The rules generated were found to be non-deterministic i.e. they could not classify any objects in decision classes. Although, the strength of rough set analysis holds for small sample size; it was not applicable for the survey data. The survey was qualitative in nature and subjective context expressed by the survey participants through phone interviews was used to analyse the co-relation between the attributes.

Overall, the survey of portal developers added a richer perspective to the information gathered via the company case study in describing the state-of-the-practice in testing portal applications and highlighting difficulties in testing. Moreover, results of the empirical assessment were strengthened by combining two methods of qualitative inquiry, case study and survey (methodological triangulation).

4.8 Summary

In this chapter, I first presented survey design, participant demographics. Next, the results of the study and its interpretations are provided. The results provide empirical evidence on the nature of challenges that impact comprehensive testing of portal applications. In addition, the results highlight requirements on testing approaches in areas where portal applications cannot be tested. The next chapter uses the study results to formulate requirements that are appropriate for addressing the difficulties articulated by the results.

Chapter 5. Portal Application Testing Process

Results of the study in Chapter 3 and 4 highlighted difficulties in automating unit and integration testing of portal applications. In this chapter², I first list the requirements for the needed testing techniques and tool support. In addition, the reasons why existing techniques are inadequate are described. Then, I explain the testing using these techniques through usage scenarios. This is followed by a discussion of how these testing techniques fit into the overall testing process of portal applications. This chapter concludes with a summary of the proposed testing process.

5.1 Requirements for Portlet Testing

Portlets are the key application building blocks of portal pages forming the controller layer of portal applications, developed using MVC. They rely on services provided by the portlet container which in turn is tightly integrated with the portal server. The portal server as well as the portlet container are black boxes from the portlet application developer's point of view and are only accessible via the portlet API. Therefore, testing portal applications in an

² A part of this chapter is published as:

Xiong, W; Bajwa, H. and Maurer, F. (2005) **WIT: A Framework for In-container Testing of Web-Portal Applications**, Proceedings of International Conference of Web Engineering (ICWE 2005) LNCS, Volume 3579, Jul 2005, Pages 87 - 97

automated way is a challenge. Effective testing approaches should satisfy the following objectives:

- 1. Test Portlet API Methods:** The portlet API provides access to container services and user information via specific objects (Appendix A.2). Some examples of these objects are portletrequest, portletresponse, portletsession objects and other application specific environment objects. In other words, the API supports interactions between the container and application code in the form of method calls to these objects. To process a client request, a portlet request object is assembled by the container using the data submitted by the browser. This object is then forwarded to the service methods (doView, doEdit, and doHelp) defined in portlet application code. The application code executes using the request and any other objects as required. As the final step, results are assembled by the container as a response object that is sent back to the client browser. The request and response objects are primarily responsible for interactions between the container and application code. The application code uses these objects accessible through the portletAPI as part of the application logic to process requests. Portlet errors may come from the container interacting incorrectly with application code or any unpredictable changes caused in the portal server environment as a result of these interactions. For instance, changed values of environment objects at runtime may create side effects on pieces of the application interacting with these objects. **Testing methods that use the context provided by the container-assembled objects requires a technique to access and modify these objects before the portlet executes and validate their state after the application code executes.**
- 2. Test Deployment Related Errors:** Execution of portal application code is sensitive to its deployment environment. As a result, developers cannot ensure error free execution when migrating portal applications between the staging and production environments. The key reason for errors during application integration stem from differences between these environments. One difference between the deployment

environments may be because the version of a specific library referred by the portlet code is different in the staging and production environment. Another source of deployment related error occurs due to incorrectly set environment attributes. These environment attributes are configured within the container at deployment time by reading the parameters from the portlet descriptor files. **Testing a portlet application to isolate errors that surface at deployment time requires an approach that supports executing portlet tests inside the container environment.**

3. **Security–Role Based Testing of Resource Access:** Access to sensitive portlet resources is controlled by assigning permissions (roles) and granting access to individual users or user groups. Without automated testing support the administrator manually verifies whether the permissions on a portlet resource have been correctly assigned after application deployment. This is time a consuming activity. **Therefore, there is a need for an automated testing framework that will allow setting up different types of users for role-based unit testing.** In addition, when an unauthorized portlet resource is accessed the tests should fail indicating incorrect invocation.

5.2 In-Container Testing of Portlets

Testing a portlet application for errors at deployment time and testing portlet API methods requires an approach where the test code executes in the “real” container environment and has the ability to access and control portlet container environment specific objects. This approach is provided by Cactus (Appendix B). Cactus can test servlets, EJBs and JSP components. However, Cactus at present does not support testing of portlets. Moreover, the in-container testing (ICT) approach implemented by the Cactus framework is restricted because components tested using Cactus are instantiated as normal classes in the test code versus using a “real container” to instantiate and manage the component’s lifecycle. Such an approach, though useful for testing some aspects of a portlet application, may not be adequate to detect deployment related errors as well as for testing portlet API methods and

its interactions (refer Section 5.1 – (1), (3)). This is because portlets depend heavily on services provided by the portal server environment (Chapter 2, section 2.2.3). Therefore, in the case of portlets it becomes imperative to test portlet API methods and its interactions using services provided by the “real container”. Testing frameworks such as httpUnit and jWebUnit (Appendix B) can be extended to support black box testing of portal application by querying the portal server externally and verifying the HTML content received by the client. However, these frameworks suffer from many drawbacks. First, they do not provide detailed control over the portal server environment; as a result constructing a test set up state is time consuming. Second, validating the response returned by the portal server is done by parsing the HTML content which is time consuming. Third, which is also the most important reason is that test methods developed using HttpUnit API submit forms and links using HTML element identifiers which in case of portlets are encoded and generated dynamically by vendor provided APIs. As a result, element identifiers change each time an application is deployed making automated functional unit testing difficult.

5.2.1 WIT Testing Framework

In order to support automated ICT of portlets, a testing tool Web Portlet In-container Testing Framework (WIT) was developed by (WIT, 2005; Wenliang Xiong et al., Jul 2005). The key idea underlying the ICT approach implemented is to intercept calls to the portlet API methods made by the portlet container via the portal server. When the method call is intercepted, it is annotated by inserting suitable instructions (test code) at appropriate points. Prior to each portlet service method call, test set up code is executed, and then the “normal” execution of the method is resumed. Thereafter, the state of environment is validated by verifying the state of objects recorded against the expected state and then reported as part of test results.

WIT provides an API to write portlet ICT test cases and integrates components that support test case execution inside the portlet container. It also supports reporting results of test execution. It uses AspectJ technology (AspectJ, 2005) to intercept calls between the

portlet API and the container by weaving the portlet source code with the test code at appropriate points. The testing process is initiated by a testing client that simulates the invocation of a portlet from a browser and assembles the portlet request. The results of the test execution get stored in the repository (implemented as an in memory database) making it possible to report results back to the testing client. As a result of executing test cases in this manner, the portal server environment can be controlled.

5.2.2 Usage Scenario of WIT

To demonstrate an ICT testing process using WIT, the *shipping portlet application* is used as an example (complete source code of the application is available in Appendix E).

This application consists of portlets that together perform the process of tracking shipments using order details related to customers. Figure 5-1 shows portlets (order details, order summary, customer details, tracking details and account details) supported in the view mode included in the shipping portlet application. The test scenarios developed in this example are specific to issues discussed in section (5.1).

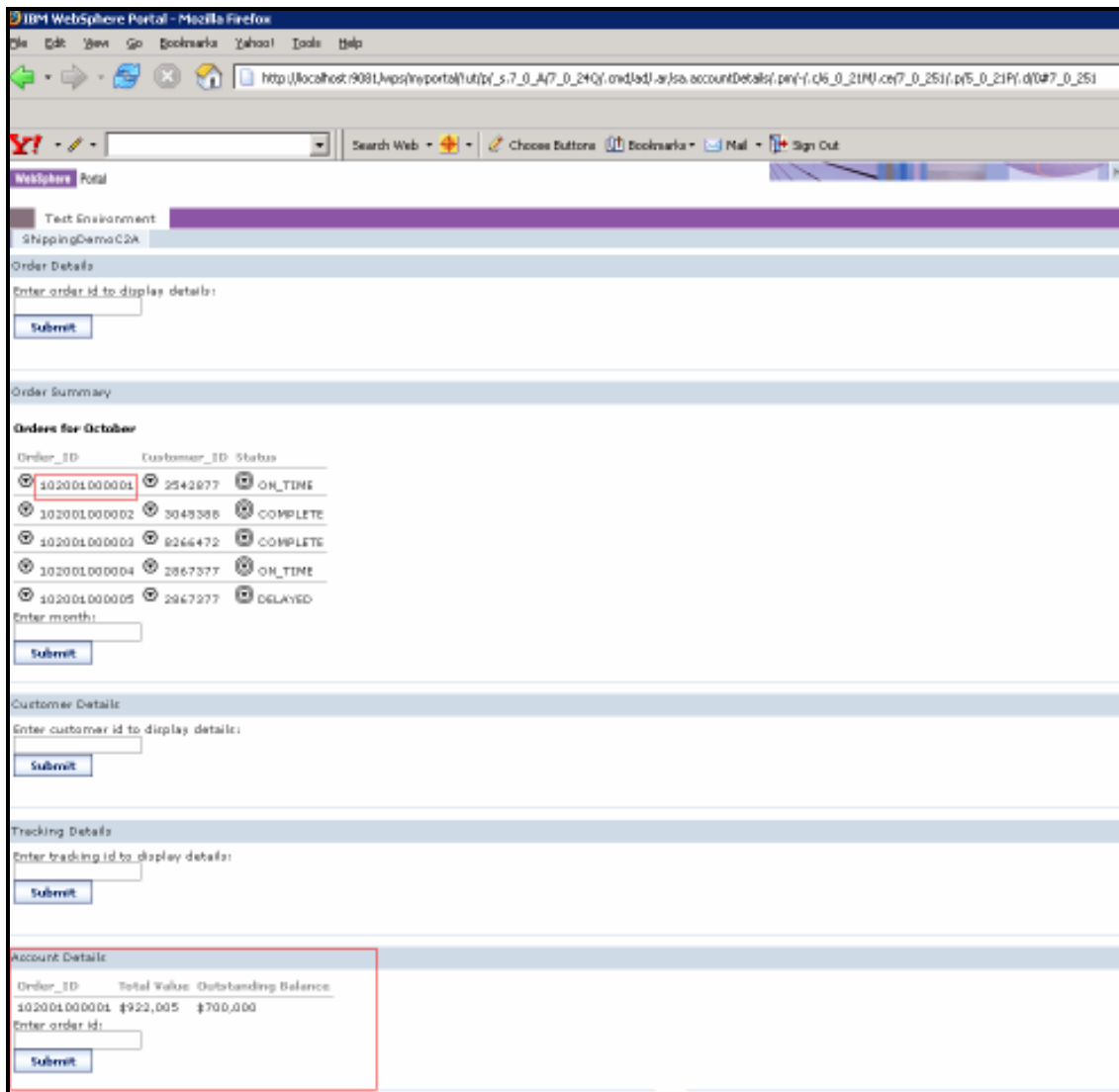


Figure 5-1: Shipping Portal Application in View Mode- Account Details Portlet.

The portal server environment used to deploy this application is IBM WebSphere portal server (WPS). Test cases have been developed for the Accounts portlet that retrieves account information such as the total value and outstanding balance related to a specific order.

Scenario1: Test Portlet API Methods

The Accounts portlet (Figure 5-2, line7) extends the PortletAdapter class which is an implementation of the portlet interface; this provides portlet API methods such as init, login, doView, destroy and logout. As shown in Figure 5-2, line 17 the class has a doView method (method associated with the portlet in the view mode). This method is invoked when the portlet is rendered in view mode.

```

1 package com.ibm.wps.portlets.shippingcsl;
2 import org.apache.jsp.wps.portlets.*;
3 import org.apache.jsp.wps.portlets.event.*;
4 import java.io.*;
5
6
7 public class AccountsPortlet extends PortletAdapter implements ActionListener
8 {
9     private static final String PREFIX = "";
10    public static final String ACTION_NAME = PREFIX + "actionName";
11    public static final String ACCOUNT_DETAILS = PREFIX + "accountDetails";
12    public static final String ACCOUNT_ID_ENTRY = PREFIX + "accountIdEntry";
13    public static final String ACCOUNT_ID = PREFIX + "accountId";
14    public static final String ACCOUNT_DETAIL_BEAN = PREFIX + "accountDetailBean";
15
16    /**
17     * @see org.apache.jsp.wps.portlets.shippingcsl.AccountsPortlet.doView
18     */
19    public void doView(PortletRequest request, PortletResponse response) throws PortletException, IOException
20    {
21        try {
22            // get the action associated by the user
23            String action = (String) request.getPortletSession().getAttribute(ACTION_NAME);
24
25            // if no java bean object is associated to the action then return details
26            AccountDetailBean adb = null;
27            request.setAttribute(ACCOUNT_DETAIL_BEAN, adb);
28
29            if (action == null || action.equals(ACCOUNT_ID_ENTRY)) {
30                // show account orgs
31                String actionURI = ShippingUtils.createActionURI(ACCOUNT_DETAILS, response);
32                adb.setActionURI(actionURI);
33                getPortletConfig().getContext().include("/WEB-INF/jsp/AccountDetailEntry.jsp", request, response);
34            } else if (action.equals(ACCOUNT_DETAILS)) {
35                String actionURI = ShippingUtils.createActionURI(ACCOUNT_DETAILS, response);
36
37                adb.setActionURI(actionURI);
38                String accountId = (String) request.getPortletSession().getAttribute(ACCOUNT_ID);
39
40                AccountDetail ad = ShippingDB.getAccountDetail(accountId);
41
42                if (ad != null) {
43                    adb.setAccountDetail(ad);
44                    // Call a context for the success session for showing the JSF that shows the user details
45                    getPortletConfig().getContext().include("/WEB-INF/jsp/AccountsView.jsp", request, response);
46                } else {
47                    // No account details were found
48                    adb.setErrorMessage("Account id " + accountId + " not found.");
49                    // Call a context for the success session for showing the JSF
50                    getPortletConfig().getContext().include("/WEB-INF/jsp/AccountDetailError.jsp", request, response);
51                }
52            }
53        } catch (Exception exc) {
54            getPortletLog().error("ServletInvokePortlet: An error occurred" + exc.getMessage());
55        }
56    }
57 }

```

Figure 5-2: doView Method - Accounts Portlet Class.

The first thing that happens in the `doView` method is that the user action is determined. Next, an instance of the account detail bean is created that will contain all the relevant data related to an order. Then, the account detail bean is set as an attribute in the portlet request object.

Account Id is retrieved from the request object and sent to the database (in this case `shippingDB` stores the account information) to get detailed information of the account. This information is returned as an account detail object; which in turn is set in the account detail bean. The account detail bean is passed from the `doView` method to the JSP (Figure 12, line 44). This is done by the `setAttribute` method on the `PortletRequest` object. As the final step, the JSP establishes a reference to the bean using a `portletAPI` tag which embeds the information in the HTML output markup.

In order to test the `doView` method of the `AccountsPortlet` class, first the `AccountsPortlet` test case class is created that extends `WITTestCase` as shown in (Figure 5-3, line 7). For each test case, a pair of methods is written *before_doView_testGetAccountDetail* and *after_doView_testGetAccountDetail*. This pair of methods follows a specific naming convention consisting of three parts. The first part is either “before” or “after”; the second part is the name of method (`doView`) being tested; and third part is any string that makes the test method name more meaningful (`testGetAccountDetail`). The method *before_doView_testGetAccountDetail* method sets up the current state of processing by setting the user action and account ID in `PortletSession` (Figure 5-2, line 22-24).

```

package
import
import
import
import
import

public class ... extends
    "accountDetails"
    "orderId"
    "accountId"

/* This method sets up the test environment before executing the method under test. */
public void

    throw

/*This method will make the portal invoke the action in doView to get the acct details.This method is
 * responsible to run a specific part of the doView method--in this case show acct details*/
    "accountId"
/*to show acct details use the required accountId */
    "112001000000"

/*We are asserting the results of executing the doView method to make sure the result is as expected. */
public void

    throw

        null
        null
        "accountDetailBean"

/* Assert 1: We are making sure that account details bean should not be null if the acct ID was correctly set-up */
    "Account detail bean is not null"

/* Assert 2: If account details bean is not null we check whether the account details for the bean is not null */
    "account details is not null"

/* Assert 3,4: Make sure that the values for the details is retrieved as expected
    "TotalValue is calculated correctly"
    "$30,000"
    "Outstanding balance is incorrect"    "$0"

```

Figure 5-3: doView Test Case - Accounts Portlet Test Class.

In the *after_doViewtest_GetAcctDetail* method, results of executing the doView method (Figure 5-3, line 28) are validated using the standard unit test methods to verify expected output and to report the success or failure of this test (Figure 5-3, lines 40-52). This test case validates that Account portlet will render valid content when deployed in the portal server environment. In addition, exceptions thrown when the Accounts portlet executes are captured and reported by the ICT tests.

Another important aspect is that in the *after_doViewtest_GetAcctDetail* method the environment state can be cleaned up. This is important because ICT tests are likely to be executed in the staging or production portal server; thus restoring the original state of the environment after test execution finishes ensures that the environment state is exactly the same before and after running the tests. This minimizes side effects on subsequent test runs resulting from unpredictable changes in the environment after executing the tests. For instance after the test case execution finishes, the database contents must be restored. Modification to the production build environment should be minor because any modification may change the environment resulting in adverse side affects on existing applications.

Scenario 2: Test Deployment Related Errors

The PortletSettings object (Appendix A.2) contains configuration parameters accessed by the portlet at runtime. This parameter is defined in the portlet descriptor file called portlet.xml. The portal administrator uses the administrative interface to configure individual portlets by editing the configuration parameters before deploying the application into the production environment. For instance, the accounts portlet (Figure 5-4, line 26-30) accesses the database connection string by reading the configuration parameter from the portlet descriptor file.


```

55  /* This method sets up the test environment before executing the method under test. */
56  public void before_actionPerformed_caseDBConnectionSetting(ActionEvent event)
57  {
58
59      PortletRequest request = event.getRequest();
60
61      //Assert: Is there any data that the request is not null
62      assertNotNull(request);
63
64  }
65  /* We are asserting the results of executing the doView method to make sure the results are as expected. */
66  public void after_actionPerformed_caseDBConnectionSetting(ActionEvent event)
67  {
68      PortletRequest request = event.getRequest();
69      String expected_dbConnectionSetting = "jdbc:db2://localhost:50000/AccountsDB";
70
71      PortletSettings settings = request.getPortletSettings();
72      PortletApplicationSettings applicationSettings = settings.getPortletApplicationSettings();
73
74      //Database connection string to read from the deployment descriptor
75      String actual_dbConnectionSetting = (String) applicationSettings.getAttribute("databaseConnectionSetting");
76
77      //Assert: Is there any data that the database string is not null
78      assertEquals(
79          "Connection Used for the Accounts Database in this environment isn't",
80          actual_dbConnectionSetting,
81          expected_dbConnectionSetting);
82
83  }

```

Figure 5-5: actionPerformed Test Case - Accounts Portlet Test Class.

Another test scenario is where the doView method depends on a service provided by the portal server. The functionality implemented in the method is using the service credential vault which enables portlets to access credentials for authentication as shown (Figure 5-6, line 31). This service class needed is activated by the portal administrator. WIT can be used to test for missing service classes that cause portlets to provide no functionality.

```

import
public class          extends          implements
private static final
public static final
public static final
public static final
public static final
public static final

/* request and response objects are provided by the container */
public void          throws

try
// get the session specified by the user

//method-level security problem to ensure method-level security
//method-level security class is provided by the portal server
.....
class

```

Figure 5-6: Credential Vault Portal Server Service.

Scenario3: Security–Role Based Testing of Resource Access

Role-based testing of resource access verifies whether permissions assigned to portlets are correct. This is explained using the scenario whereby a portal end user David, is accessing Accounts portlet which ideally he should not have access to. Testing for this scenario is set up by specifying a user role and password in the test configuration file (Figure 5-7, line 13-14). WIT allows setting up a series of user roles for accessing different portlet resources in the test configuration file. This information is used for authenticating David prior to executing the request for the portlet service method (doView). If the request is successful, the doView method in AccountsPortlet will be invoked which should not be accessed if permissions were assigned correctly. Thus, the security test case indicates a failure, when the doView method is invoked.

```

1 <testSuite xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="WIT-Config.xsd">
2   <controllerServletParams>
3     <!-- in seconds for single portlet -->
4     <singlePortletRuntime>30</singlePortletRuntime>
5     <!-- in seconds for all portlets -->
6     <allPortletsRuntime>1000</allPortletsRuntime>
7   </controllerServletParams>
8
9
10  <testSuite id="1">
11    <testSuiteName>TestSuiteAuthenticationTestForAccountsPortlet</testSuiteName>
12    <authentication>
13      <accountFieldname[loginForm="userId">John/</accountFieldname>
14      <passwordFieldname[loginForm="password">Does/Passwords</passwordFieldname>
15      <authURL>https://localhost:9081/wps/portal/!ut/p/.and/1fn/authURL</authURL>
16    </authentication>
17
18    <testInvocations>
19      <testInvocation invokerURL="http://localhost:9081/wps/portal/!ut/p/.and/1fn/LoggedIn">
20        <testCaseMethod>
21          <originalClassName>com.ibm.wps.portlets.shipping2a.AccountsPortlet</originalClassName>
22          <originalMethodName>doView</originalMethodName>
23          <testClassName>com.ibm.wps.portlets.shipping2a.AccountsPortletTest</testClassName>
24          <testMethodName>doView.testGetAccountDetail</testMethodName>
25        </testCaseMethod>
26      </testInvocation>
27    </testInvocations>
28  </testSuite>
29 </testSuite>
30 </testSuite>

```

Figure 5-7: Snippet of WIT Test Configuration XML File.

Test Case information is specified in an XML based configuration file as part of the test execution set up. In addition, a properties file that specifies the location of the portal application and test source code must also be provided. The sample configuration file and properties file are attached in Appendix E.

Test Cases are executed after writing ICT tests by invoking a custom (ANT) command that compiles, deploys the test cases and invokes all of the tests. At the end of the script run, test results are displayed (Figure 5-8) in the script window which shows the results of executing ICT test methods (Figure 5-3 and Figure 5-5). An Ant based script can be integrated with the regular automated deployment and build process.

```

C:\PortalTesting-June24\WITInvokerClientForShippingDemoC2A>ant -f build-wps.xml
Buildfile: build-wps.xml

WIT:
testCase2Aspect:
weaving:
deploy:
  [move] Moving 2 files to C:\PortalTesting-June24\ShippingDemoC2A\WebContent\WEB-INF\classes
start2Test:
[java] Starting to execute Tests.....
[java] Test Result(s):
[java] -----The failed test method-----
[java] [WIT] Test Suite Name: null
[java] [WIT] Test Run ID: 1247977
[java] [WIT] Original Class Name: com.ibm.wps.portlets.shippingc2a.AccountsPortlet
[java] [WIT] Original Method Name: actionPerformed
[java] [WIT] Test Class Name: com.ibm.wps.portlets.shippingc2a.AccountsPortletTest
[java] [WIT] Test Method Name: actionPerformed_testDBConnectionString
[java] [WIT] Test Result: Connection Used for the Accounts Database in this environment is
[java]   expected:<jdbc:db2://localhost:50000/AccountDB> but was:<jdbc:db2://localhost:20000/AccountDB>
[java] -----Test Run Statistics-----
[java] Total number of test methods that were Expected to Run: 3
[java] Total number of test methods Actually Run: 2
[java] Total number of test methods Successfully Run: 1
[java] Total time taken to run the tests in seconds is: 2.0

BUILD SUCCESSFUL
Total time: 7 seconds

```

Figure 5-8: Test Execution Results using WIT.

5.3 Portlet Testing using Mock Objects

Portlet testing using mock objects is a unit testing strategy for portlets, referred sometimes as out-of-container testing technique. Mock objects (MO) for portlets need to provide objects that simulate the portlet container. In other words, mock implementation of the portletAPI objects such as portletRequest, portletResponse, portletSession must be provided. These objects create the context in the test set up methods for executing the portlet test methods. Portlet testing, using mock objects is a contrasting approach to ICT described above. This is because unit tests using MOs execute in the “*simulated*” container versus the “*real*” container.

(PortletUnit) is a framework built for testing JSR 168 portlets by extending two open source projects. One is the (ServletUnit) framework and the other is (Pluto), which is the basic reference implementation of the portletAPI (Appendix B). Pluto is embedded as the portlet container in the PortletUnit testing framework for executing the portlet code. The testing framework provides an interface to access the portlet directly. Furthermore, the

framework has APIs that allow access to MO objects to initialize the specific request as well as to validate the current state of processing.

The two key benefits of using MO (described in section 2.5.2, Chapter 2) are reduced test execution time and rapid test feedback. This allows effective TDD process for portlet applications because the effectiveness of TDD is inversely proportional to the execution time of the tests. However, as MOs are different from the “*real*” objects they replace, they do not assure that the portlet methods under test will run correctly when deployed in the real production portal server environment.

5.4 Portal Application Testing Process

Portal applications can be tested by combining unit testing with portletUnit and ICT with WIT in three different environments. The three different environments are development and unit test, staging, and production (section 2.4, Chapter 2). Figure 5-9 shows this based on the type of development and deployment environment. In addition, Figure 5-9 illustrates which kind of tests should run, where and how each of the described testing techniques fit into the overall testing process of portal applications. This answers the specific research questions outlined in Chapter 1, section 1.5.

5.4.1 Unit Test Environment Level Tests

The first testing approach in the testing process (Figure 5-9, 1.1, 1.2 and 1.3) is testing in the development environment. The model layer business logic is unit tested first in a comprehensive manner, in this environment using JUnit. This is an important step to ensure that portlets work correctly because portlets use services provided by the model layer. Next, the developer implements portlet code and writes unit tests using the portletUnit API. However, writing unit tests in this manner may be more appropriate for portlets that have complex business logic for example logic related to user preferences and the invocation of several model layer methods. In addition, portlets may have complex dependencies on

external infrastructure (registry services, content management systems) which must be mocked accordingly when testing is performed in this environment.

Portlet and the model layer unit tests can be maintained as part of a common test suite. Thus, model and portlet layer can be tested continuously. The model of development is making small changes to code, building it and executing the model and portlet unit tests each time. This cycle can be repeated several times per day. In addition, this can be done as part of a fast compile and regression testing strategy using JUnit test runner client. Once this cycle is complete, the application code is deployed in the portlet toolkit environment for functional unit testing. However, the number of times deployment happens in this environment should be minimized. For example it can be deployed once an hour to save development time because initializing the environment and application deployment is time consuming. The benefit of conducting testing along these lines is that significant overhead of testing portlet changes, when portlets are implemented can be reduced. Consequently, this reduces the overall development, deployment and test cycle of portlets.

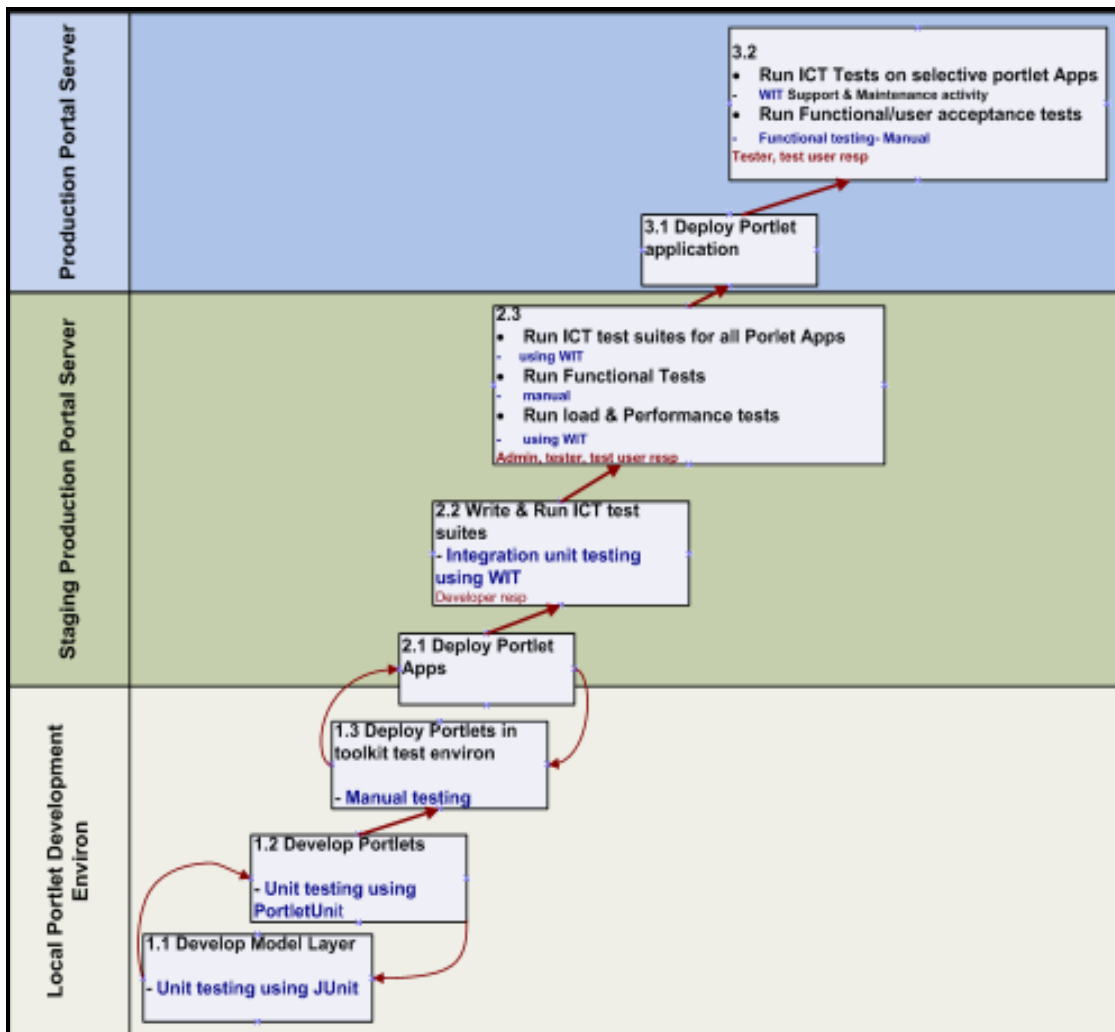


Figure 5-9: Portal Application Testing Process- testing activities in different run-time environments.

5.4.2 Staging Environment Level Tests

The second approach in the testing process (Figure 5-9, 2.1, 2.2, and 2.3) is during application deployment when the application is migrated to the staging portal server; which is an environment that mirrors the production portal server. This kind of testing merges into the **integration unit testing process** of portal applications. In addition, this type of test execution may be integrated with automated build and deployment process promoting continuous integration testing; which is executing integration tests at regular intervals

during portal application development. Testing in this environment is based on who (developer, administrator, tester or end user) executes the tests. First, developers are responsible for writing ICT test cases using WIT for their individual applications and executing tests at regular intervals. The number of times ICT tests must run varies and can be run every hour or once in a day. Because ICT testing is slow, executing tests frequently is infeasible. Next, ICT test suites must be run, when portlet applications are integrated by the administrator in the staging environment. This will detect any errors as a result of side effects in the deployment environment when different portlet applications are integrated. The suggested integration unit testing process should to a certain extent replace the need for automated functional unit testing.

After the application is successfully deployed in the staging environment the functional testing of the portal application (jsp layer) is important to ensure that portal pages are rendered correctly. This is performed by a walk through of functional requirements by testers or end users. It is important to note that manual testing can never be completely removed from the testing process because it may reveal bugs not detected by the automated tests. However, automated ICT of portlets will reduce manual functional testing effort because errors related to portal application integration are detected early - prior to release in the production portal server.

Automating the deployment process may not be necessary for executing unit tests using PortletUnit because the tests are executed out-of-container. As a result, deploying the code is not needed. In contrast, ICT requires application code to be redeployed before performing testing by removing the deployed code from the environment and redeploying (hot deploy). The time taken for a hot deploy is dependent on the vendor specific implementation of the portal server. The deployment process may be automated for ICT. This may be done by using different command-line tools together with (ANT) for application deployment. Without that, continuous integration testing of portal applications is not possible.

5.4.3 Production Environment Level Tests

The third testing approach, in the testing process (Figure 5-9, 3.1, 3.2) is when the portal application is migrated to the production portal server. It is recommended that minimum automated testing be conducted in the production portal server. Any side effect caused as a result of test case execution may change the production database contents and requires to be rolled back correctly. Moreover, the modification to the production build environment should be minor. For instance, adding test suites, automated tool related classes may result in modifications to the production environment. This may cause adverse effects on existing portal applications. In order to reduce the possibility of deployment related errors, it is recommended that staging and production portal server environments and its backend components are replicated as an *exact* mirror of each other. As part of portal application support related activities, a subset of portlet ICT test cases can be run over regular intervals. In other words, test case execution can be limited to testing certain key portlets. The result of executing these tests is to check availability of portlets in production and their performance over a period of time.

5.5 Summary

Portlets are key application components of portals. Therefore, it is important to test them in a comprehensive manner. Two complimentary testing techniques, the mock object approach using PortletUnit and in-container testing using WIT are discussed in this chapter. In-container testing focuses specifically on detecting environment specific portal application errors when the application is integrated with the deployment environment. Although the in-container testing approach is important, it is not feasible as a fast unit testing strategy because of the overhead in initializing the portal server environment which makes test execution slow. On the other hand, unit testing in the simulated container using a mock object approach can support fast unit testing. However, it cannot ensure success of an application at deployment time because the tests execute using the context provided by

the simulated container. By outlining the portal application testing process, I have demonstrated how both these complimentary strategies can be integrated into a portal application development process. Although the testing techniques are complimentary in nature, they have trade offs. Therefore, for testing certain types of portal application the benefit of using one testing technique may outweigh the other. Accordingly, each of testing techniques must be evaluated before using them. In the next chapter, I will discuss results of exploratory study conducted to assess the viability of the proposed testing approaches in the industry.

Chapter 6. Empirical Evaluation

The previous chapter described techniques for automated testing of portal applications together with how these techniques are integrated into the overall testing process. In this chapter, I first present objectives of the exploratory study conducted to assess the perceived viability of the suggested testing techniques and their likely future usage in an enterprise. Next, I detail the study methodology, participants and results. This chapter concludes with a brief summary.

6.1 Selection of the Methodology

In order to empirically validate the testing process and practices discussed in this thesis, a longitudinal study investigating aspects of process improvement should be conducted. However, certain factors limited a longitudinal study during this research. First, our industrial partner Sandbox Systems was a consultant company and its portal developers worked as consultants for different companies on offsite portlet application development projects. Conducting a study in the consultant company requires willingness of the company to allow use of their environment. However, Sandbox and its consultant company environments were unavailable for accessing and deploying portal applications. Consequently, I conducted a short exploratory study as the first step for evaluating the viability of the testing techniques and perceived usefulness of the tools.

6.2 Objectives

The objectives of the study are:

1. To evaluate the perceived usefulness of the proposed portal application testing techniques.
2. To evaluate the likely future usage and viability of the testing techniques.
3. To provide feedback on how developers anticipate testing techniques to be integrated into a portal development environment.

The main research questions are:

Question 1: To what extent do the participants perceive the testing techniques as useful?

Question 2: How likely are the participants to use the testing techniques in their environment?

Question 3: How do the participants perceive the testing techniques should be integrated into the development, test and deployment process of portal applications?

The first question evaluates the usefulness of the testing techniques (objective 1). The second question evaluates whether the approaches will be used (objective 2) and the third question identifies how these techniques can be incorporated into an existing development, test and deployment process (objective 3).

6.3 Study Methodology and Participants

Prior to conducting the study ethics approval was obtained from the University of Calgary (Appendix D). The participants also completed and returned informed consent forms included in Appendix E.2. A pilot study was first conducted which formed the basis of refining the study methodology presented. Details of the pilot study are included in Appendix E.1.

The study was conducted with 15 participants having different levels of experience (4mths-2yrs) in portal technology. 12 participants were working for a company developing

portal applications by integrating backend legacy systems (Domino Lotus Notes, SAP) using a portlet front end. These participants were involved in a range of activities from designing portal applications, developing the business interface layer (portlet services layer), developing portlets and deploying and administering the portal applications in WPS. The portal server deployment environment used by the company was WPS. Five out of these 12 participants had previously participated in the pilot study conducted at the university. Therefore, they had prior knowledge of the testing techniques and had used WIT to develop simple ICT test cases.

Two participants out of 15 were using portal technology to integrate existing Web applications for their respective companies. 1 participant had built portlet applications in the past over a 4 month training period.

The participants were given a presentation (Appendix G) for approximately 35 minutes introducing the goals of ICT and MO techniques for portlets. This was followed by demonstrating how a test case scenario was written and executed. The presentation was followed by a discussion that provided feedback on the feasibility of the discussed approaches. The participants were asked to complete the questionnaire at the end of the discussion. All 15 participants completed responses for the questionnaire.

6.4 Results

The study was conducted to explore the perceived usefulness of the proposed testing techniques as well as how these techniques can be integrated into the existing testing process. The insight to these high level questions can be found by looking at answers to the specific research questions.

Question 1: To what extent do the participants perceive the testing techniques as useful?

When asked to rate the perceived usefulness of ICT using WIT (Table 6-1) 67% of the participants indicated that it would be helpful in detecting deployment related errors. The

nature of deployment errors described by one participant was related to missing and incorrect versions of class libraries between local development and staging portal server environments.

Table 6-1: Perceived Usefulness of WIT.

Perceived Usefulness Participants (n=15)	Portlet Deployment Related Errors	Interaction between Container & Application Code (Portlet API)	Security Role based Resource Access
Not at all to Very Little	2 (13%)	4 (26%)	0 (0%)
Average	3 (20%)	5 (33%)	6 (40%)
Helpful to Very Helpful	10 (67%)	6 (40%)	9 (60%)

In terms of detecting errors as a result of the interaction between container and application code 40% rated the approach to be helpful whereas 33% rated it as average and 26% reported it as not very useful. One of the reported reasons, for limited usefulness was that portlets were being developed using frameworks such as JavaServer Faces (JSF). These frameworks worked as wrappers abstracting the core portlet interface methods, and the portlet service method (doView, doEdit) source code was unavailable. In the case when an error in executing the portlet occurred, the developer was confronted with a black box framework level interaction. WIT in its current state cannot be used to test framework level methods and their interactions. This provides motivation for WIT to explore testing of methods and its interactions supported by the portlet development frameworks versus the current portlet service methods (doView, doEdit etc).

60% of the responses indicated that the approach would be very helpful in testing role based resource access. One respondent indicated that WIT was helpful because it allowed setting up a series of user roles so, *“you can test the context of what the portlet is about to do; based on who you are”*. -- Developer

Although the participants provided positive responses on the usefulness of the ICT approach; they suggested usability specific improvements of WIT. The results in (Table 6-2) show the level of perceived difficulty by the participants in writing and running ICT tests using WIT.

Table 6-2: Perceived Usability of WIT.

Perceived Usability Participants (n=15)	Writing WIT Tests	Running WIT Tests
Hard to Very Hard	9 (60%)	10 (66%)
Average	4 (26%)	2 (13%)
Easy to Very Easy	2 (13%)	3 (20%)

60% of responses indicated that writing test cases using WIT was difficult. 66% responded that running ICT test cases was a lot of effort because it required editing multiple configuration files. The lack of a graphical user interface and a plug-in based IDE support was one of the frequently mentioned reasons for low usability of WIT. In one respondent's opinion the effort spent in configuring ICT test cases may act as a deterrent and reduce its usefulness because

“Strictly WIT as a developer's tool is pretty hard to use because it comes with fairly low level configuration and the need to edit multiple files. There is too much margin for error. WIT is supposed to help write better code but one may end up spending too much time configuring WIT”. -- Developer

One suggestion to improve WIT was to integrate the tool with industry tool sets by making it available as a Web tools compliant eclipse plug-in. Therefore, WIT would become usable for multitude of portlet containers providing a more generic testing approach.

Question 2: How likely are the participants to use the testing techniques (ICT and MO) in their environment?

When asked how likely the participants were to use ICT with WIT in the future in their company environment; 60% of the participants indicated a positive choice of likely to very likely. The results in Figure 6-1 also indicate the viability of this approach in the industry. One participant reported in response this questions that,

“ICT approach is sound because container itself is where a lot of the pitfalls lie in testing web context. Testing ICT you gain some understanding of what the deployment process will be like. -- Developer

Although the ICT testing approach was viable, the participants indicated that its likely usage was subject to the availability of a user friendly tool integrated as a plug-in within an IDE.

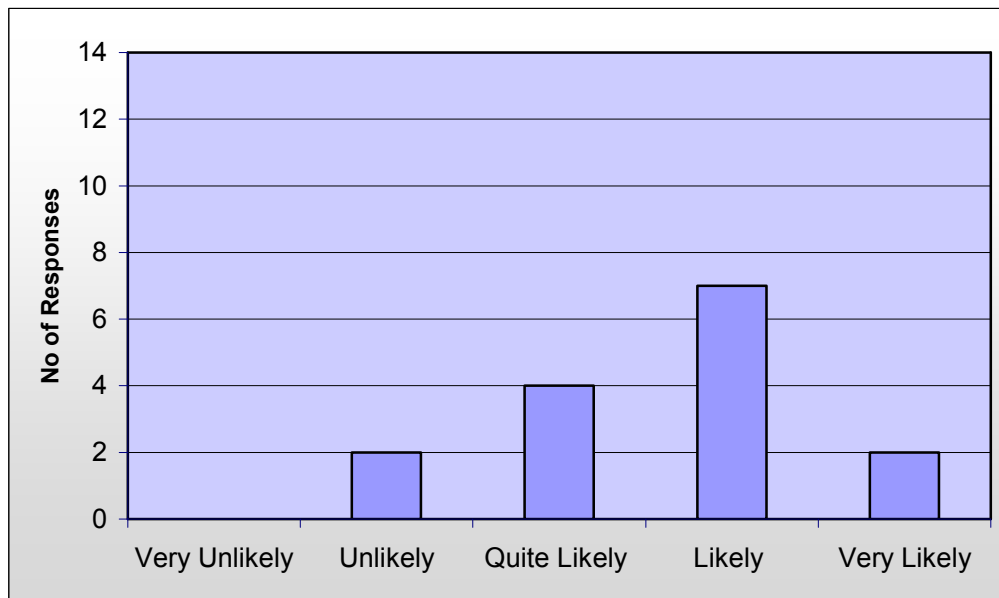


Figure 6-1: Responses showing likely future Usage of in-container testing approach using WIT.

In response to the likelihood of using MO portlet testing (Figure 6-2) approach in the future; only 46% indicated a positive choice. In one developer’s opinion a mocked portlet

API would be very useful for testing inter-portlet communication by generating mocked portlet messages and testing the behaviour of the portlet in response to these mocked messages.

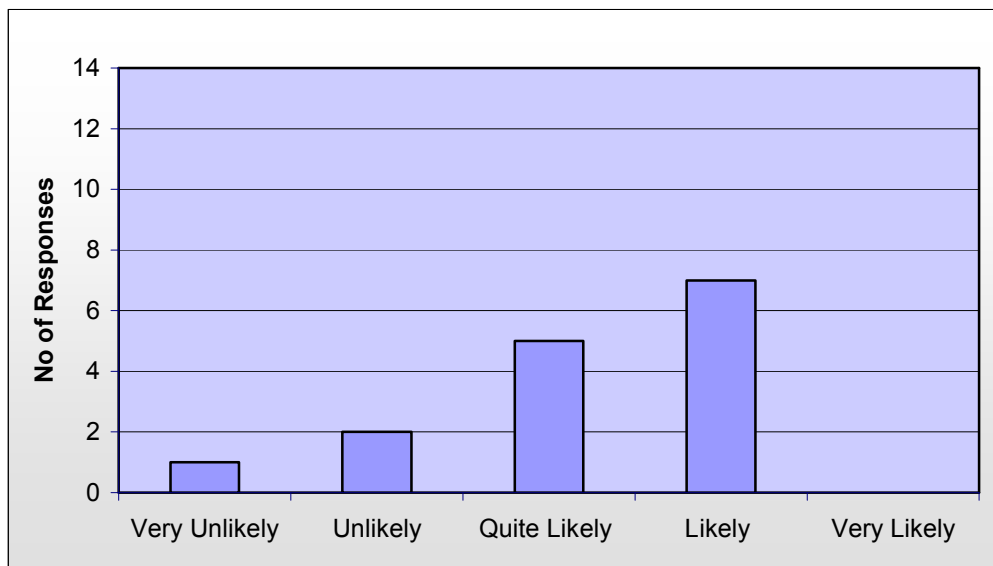


Figure 6-2: Responses showing likely Future Usage of Mock Objects.

20% of the participants provided a negative response. One of the reasons provided by one participant was the heavy dependency of portlets on the services provided by the container. Therefore, in the participant's opinion, the effort to write and execute tests outside the container was not likely to be of much benefit. In addition, it was indicated that writing out-of-container tests was suitable for portlets that implemented functionality that had less dependency on the portal server provided services. Another reason indicated was that the usefulness of this approach could only be assessed depending on the extent to which the portlet API methods were mocked and the functionality of the portlet method under test.

Question 3: How do participants perceive testing techniques to be integrated into the development, test and deployment process of portlet applications?

This question is divided further into sub questions to understand how participants perceive the integration of the testing techniques into existing testing process.

The first question inquired who is likely to write and run ICT tests. The majority of the responses (Table 6-3) indicated that ICT tests should be written by portal developers and the other 33% indicated that both developers and administrators should write the tests. (54%) Tests should be executed by portal developers as well as portal administrators depending on the accessibility of the portal server environment. A single respondent indicated that teams that support and maintain portal applications after they are released into production can execute ICT at regular intervals to ensure that portlets are available and work as required.

Table 6-3: Who should write and Execute ICT Tests.

	Writing ICT tests Participants (n=15)	Running ICT tests Participants (n=15)
Portal Developers	10 (66%)	3 (20%)
Portal Administrators	0 (0%)	3 (20%)
Developers & Administrators	5 (33%)	8 (53%)
Portal Support & Maintenance Teams	0 (0%)	1 (7%)

The second question inquired at what stage (when and where) of the development process writing and running ICT tests is likely to be beneficial. The results (Table 6-4) include responses that indicate all the three types of environment namely development, staging and production were feasible for running ICT tests. 78% of the participants indicated that ICT tests should be executed in the staging portal server environment (refer

section 2.4) for testing whether the deployed portlets work correctly. They perceived that this would help them understand the deployment process and plan production level deployment properly. One respondent indicated that ICT would delay the development process; though in their opinion it was important to execute portlet application code through ICT test scenarios closer to the end of iteration. This was indicated by the respondent that,

“MO approach is faster but not necessarily as comprehensive. From a development standpoint, I think it would slow the development process but when you get closer to end of the sprint and when you are prepared to deploy it is important to start running code through ICT scenarios. Though ICT is not viable for TDD in my opinion”.

-- Developer

59% perceived that ICT tests could be executed in the production portal server. One possible use indicated was conducting daily health checks of key portlets. Another response indicated that ICT was useful for performing preliminary portlet functionality compliance tests.

Table 6-4: Type of Portal Sever Environment for Running ICT Tests.

Type of Environment for Running ICT Tests	Participants (n=13)
Development	0 (0%)
Staging (Pre-production)	5 (39%)
Production	3 (20%)
Staging & Production	5 (39%)

The third sub question inquired how the MO approach using PortletUnit can be integrated into the current process. One participant perceived that the MO approach can be incorporated with JUnit as fine grained portlet testing for development purposes. According

to this participant, Out-of-container testing would be conducted on the local developer environment to validate existing code. After ensuring the portlet code works on the local environment, the application code can be tested on the staging environment. It was also added that ICT test cases and the MO test cases should be reusable and easily refactored. This would save considerable effort in writing test cases for different tools.

Another participant provided suggestions on how portlet MO can be used to test inter-portlet communication when developing custom portlets that did not leverage the proprietary portlet messaging frameworks. According to this developer, MO can be enabled to simulate portlet messages by referencing portlet request and response objects. This would help in understanding whether the portlets were communicating correctly with one another and behaving as expected on receiving the simulated messages. In other words, the outcome of a portlet that depends on receiving messages from another portlet can be validated. By mocking portlet messages, different scenarios within a workflow can be tested by setting up the context for testing in an automated way. This is in contrast to testing these scenarios, by creating and forwarding the messages manually and clicking on the portlet URLs. This results in testing as a time consuming activity. In the developer's opinion,

“Testing this (workflow portlets) out of a mocked container level would be very fast versus in-container sitting at the screen; clicking and working through the functionality”. --Developer

6.5 Interpretation

The study and results reported in Section 6.4 provide an indication of the perceived usefulness of the proposed testing techniques and tools although further research involving a non-prototype tool is needed. Positive responses indicate the willingness of the industry to incorporate these techniques into the testing process in the future. However, for these testing techniques to be practiced by the industry, additional factors must be addressed for

example tool design, development process and deployment process used. Participant comments related to improving the usability of WIT were highlighted.

Listed below are some limitations and improvements for future work on WIT:

1. **Lack of a Generic Portlet Testing Framework:** The WIT framework is not a generic testing framework and was designed towards executing ICT tests specifically in WPS, a proprietary framework provided by IBM. The major issue in providing a generic automated portlet testing framework is lack of standards between various portal framework providers in its current state. A generic testing framework would need to deal with dissimilarities amongst the portal server run time environments from different providers. For instance, portlet URL encoding and access is different across portal frameworks. Another issue is the use of non-standard vendor provided portlet APIs that poses additional problems for portlet testing. Unless portlet development using the standard JSR 168 API is imposed, providing a generic testing framework will remain a challenge.
2. **Source Code Dependency:** Portal framework vendors provide custom portlets. Applications are built using these custom portlets. Consequently, proprietary portal framework considerations prohibit portlet source code availability. The WIT framework annotates portlet source code by inserting instructions before and after execution of the portlet source code. To overcome this challenge WIT should support class (bytecode) level annotation.
3. **Reducing Test Effort:** WIT must evolve into an IDE plugin for the ICT approach to become an integral part of portlet testing and development process. Such a support would allow wizards for creating, writing, configuring and executing the ICT test cases with reduced effort.
4. **Informative Test Execution Results:** In order to track more information on the lifecycle of portlets (for instance, how the portlets are instantiated, state of the portlets in use), existing tools (WILEY, IBM proprietary) may be integrated with WIT. This

integration would allow more in depth information on the state of the environment and sequence of method calls invoked during portal application execution. Consequently, the test results can provide more problem-solving information if a portlet test fails. In other words, when a failure is discovered, information revealed by the test results should provide insight into the cause of failure.

A key limitation of this analysis is that the participants' common to both the survey and post study empirical study conducted is very small (five). Using these five data points and their values to establish a co-relation between the survey participants and the post study variables does not provide statistically valid results.

6.6 Anecdotal Evidence

ICT was published on a leading portlet community group (PortletCommunity) (Figure 6-3) as the top feature of week (21st- 26th, March 2005). Members of this forum are portlet developers and technical experts working on portal related technologies. Positive feedback was received from this community. This provides anecdotal evidence that the ICT using WIT for portlet testing is perceived useful for portal application developers.

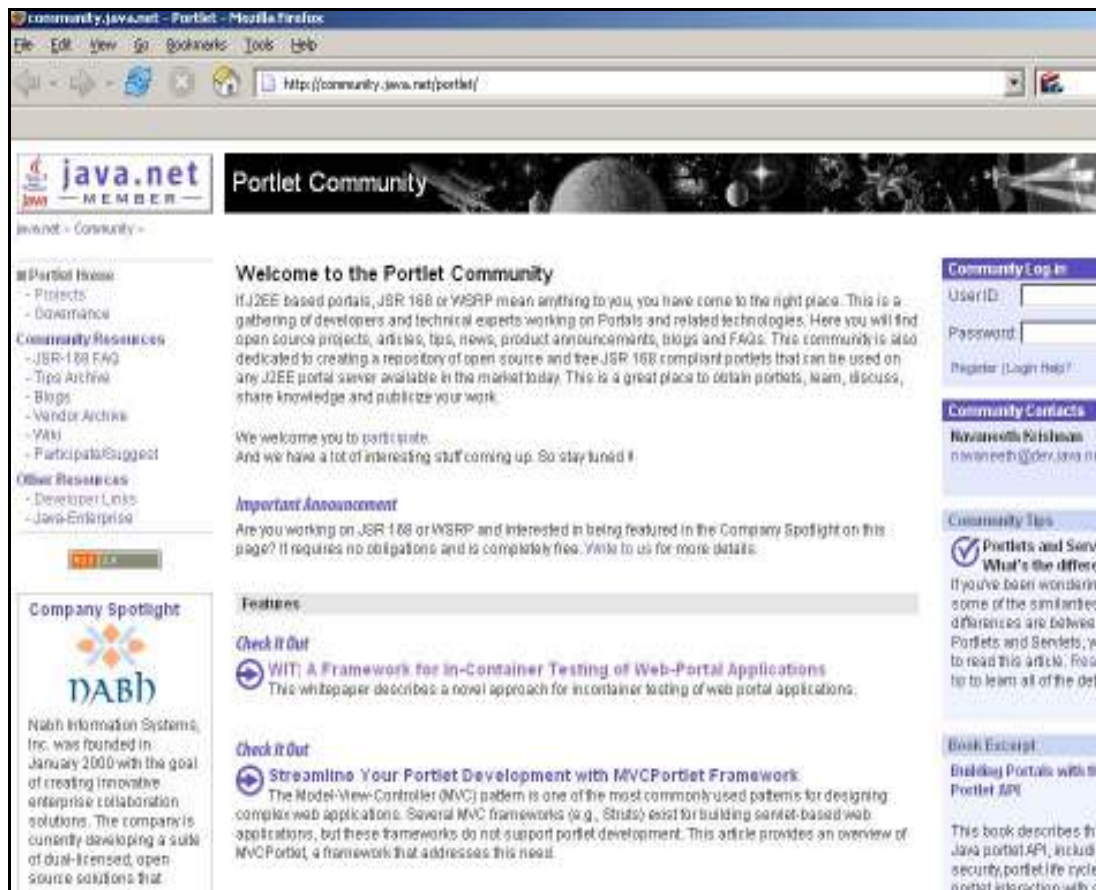


Figure 6-3: WIT as the top Feature of the week on the Portlet Community Website.

6.7 Summary

In this chapter, I reported the results of an exploratory study conducted to assess the viability of ICT and MO for portlet testing. These results provide empirical evidence that the proposed testing techniques are feasible in the industry. In particular these results may form the foundation for further research in testing of portal applications.

Chapter 7. Conclusions

I conclude this thesis by summarizing the research contributions in the area of testing portal applications. First, I reiterate the thesis problems from Chapter 1. Next, I describe my research contributions, by outlining how I solved each thesis goal from Chapter 1. Then, I suggest areas of future work in testing of portal applications.

7.1 Research Problems

Following research problems were outlined in the thesis:

1. It is not known how a portal application is being tested in the industry (state-of-the-practice) and what difficulties exist that hinder automated testing of portal applications.
2. It is not known what testing tools and techniques exist that are appropriate for testing a portal application; if there is a need to extend these techniques and develop practices for portal application testing process.

7.2 Thesis Contributions

This thesis makes the following research contributions by solving each of the problems outlined above:

1. **For this thesis a case study and survey were performed to understand state-of-the-practice in testing portal applications.** The results of the case study revealed problems with integration and unit testing of portal applications. The case study highlighted that portlets worked correctly in the test environment but errors occurred when the portlets were deployed and executed in the portal server production environment. This was reported as a severe problem as no functionality was available

to the end user since the portlet did not display any data. Furthermore, developers spent a lot of time and effort to diagnose and fix portlet application errors. As a result, the time taken to develop, test and deploy portlet applications increased. The survey results showed that unit and functional testing of portlets was conducted manually. Both the case study and the survey highlighted the need for automated testing tool for testing portlets. These results together identified the requirements that formed the basis for developing testing techniques specific to portal applications.

2. **Based on the results from the study, survey and literature review, testing techniques were developed.** Two testing techniques, the mock object approach using PortletUnit and the in-container testing using WIT were described. In-container testing focuses specifically on detecting environment specific portal application errors when the application is deployed. Although the in-container testing approach is important, it is not feasible for TDD which implies frequent test execution during development. This is because the overhead of initializing the portal server environment makes the test execution slow. On the other hand, unit testing using the mock object approach supports TDD because the tests are executed outside of the container. However, it cannot ensure successful deployment of an application in production portal server because the tests execute using a simulated container. By outlining the portal application testing process, I have demonstrated how both these complimentary strategies can be integrated into a portal application development process. Although the testing techniques are complimentary in nature, they each have trade offs. Therefore, for testing a given portal application the benefit of using one testing technique may outweigh the other. Accordingly, each of the testing techniques must be evaluated before using them.
3. **An exploratory pilot study was conducted with industry participants to evaluate the viability of proposed in-container testing with WIT and mock object testing approach based on three factors.** The first factor evaluated the usefulness and the second factor assessed the likely future usage of the testing techniques. The third

factor identified how both testing techniques are likely to be integrated into an existing development process from the perspective of the industry. The results indicated that in-container testing with WIT was useful, especially in signifying errors related to deployment of portal applications as well as for role based testing of portlets. Also, positive responses were received for the likely future usage of WIT and the willingness to adopt this testing technique. On the other hand, very few responses indicated a likely use of the mock object approach because of the heavy dependency of portlets on the services provided by the container. Therefore, the effort to write and execute tests using portletUnit would need further assessment. However, for these testing techniques to be practiced by the industry, additional factors must be addressed. For example, the tool design, development process and deployment process used needs to be improved. The prototype tool WIT should be redesigned to fix the usability issues and further research studies involving a non-prototype tool are needed.

7.3 Future Work and Conclusion

This work described the process of testing portal applications. Prior to this thesis, there was a lack of prescribed practices for testing applications developed using portal technology. It is important that such recommendations be provided to develop quality applications using portal technology. However, it is not objectively confirmed if the suggested testing techniques will have tangible and long term benefits in improving the quality of portal applications. Early empirical results validated the perceived usefulness of testing techniques provided in this thesis and point to the feasibility of techniques and usefulness of WIT prototype tool. As the next step, the proposed portal application testing process should be implemented in the industry and evaluated formally. Such an evaluation should investigate and measure specific aspects of process improvement. In addition, the cost benefit analysis of automating certain stages of the portal application testing process must be conducted.

WIT is a proof-of-concept implementation validating the in-container testing approach of portlets. Although the conceptual approach received a positive response, the tool should be extended to make it viable for use in the industry. Therefore, much work should be done before promoting this tool for use in the industry. The key conceptual idea underlying the in-container testing can be extended to support generic components. Thus, this thesis has a broader significance in testing of component based systems.

References

Abran, A., Bourque, P., Dupuis, R. and Moore, J. W. (2001). **Guide to the Software Engineering Body of Knowledge - SWEBOK**, *IEEE Press*.

ANT, Apache Java Build Tool, <http://ant.apache.org/>, last accessed, August 14, 2005

AspectJ, AspectJ Eclipse Project, Online: <http://eclipse.org/aspectj/>, Accessed: July 14, 2005, <http://java.sun.com/j2ee>, last accessed,

Astels, D. (2003). **Test Driven Development: A practical Guide**, *Prentice Hall, Upper Saddle River, NJ*.

Beck, K. (2000). **Extreme Programming Explained: Embrace Change**, *Addison Wesley*.

Binder, R. V. (2000). **Testing Object-Oriented Systems: Models, Patterns and Tools**, *Addison Wesley*.

Cactus, Cactus Server Side Testing Framework (Apache Jakarta Project), <http://jakarta.apache.org/cactus/>, last accessed, August 14, 2005

Canoo, Client Side Testing of Web applications using Canoo, <http://webtest.canoo.com/>, last accessed, August 14, 2005

Cechich, A., Piattini, M. and Vallecillo, A. (2003). **Assessing Component-Based Systems**. *Component Based Software Quality*, Springer-Verlag.

Clover, Code Coverage Tool, <http://www.cenqua.com/clover/>, last accessed, September 14, 2005

CruiseControl, Framework for Continuous Build Process, <http://cruisecontrol.sourceforge.net/>, last accessed, August 14, 2005

Culbertson, R., Brown, C. and Cobb, G. (2001). **Rapid Testing**, *Prentice Hall PTR*.

Cutter-Consortium, Poor Project Management Number-one Problem of out sourced E-projects, *Cutter Research Briefs*, <http://www.cutter.com/research/2000/crb001107.html>, last accessed, August 14, 2005

E.Gamma, R.Helm, Johnson, R. and Vlissides, J. (1995). Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley.

Fetterman, D. M. (1989). Ethnography: Step by Step. Applied Research Methods. C. S. P. Newbury Park. **Vol 17**: page 51-52.

Fowler, M. and Foemmel, M., Continuous Integration, <http://www.martinfowler.com/articles/continuousIntegration.html>, last accessed, August 14, 2005

Hepper, S., Understanding the Java Portlet Specification, http://www.ibm.com/developerworks/websphere/library/techarticles/0312_hepper/hepper.html, last accessed, August 14, 2005

Hepper, S. and Lamb, M., Best Practices: Developing Portlets using JSR 168, ftp://ftp.software.ibm.com/software/dw/wes/pdf/0403_hepper-JSR168_BestPractices.pdf, last accessed, August 14, 2005

Hepper, S. and Lamb, M. (2004). Best Practices: Developing Portlets using JSR 168 , Online: ftp://ftp.software.ibm.com/software/dw/wes/pdf/0403_hepper-JSR168_BestPractices.pdf, Accessed: August 14,2005.

HtmlUnit, Client Side Testing of web applications using HtmlUnit,, <http://htmlunit.sourceforge.net/>, last accessed, August 20, 2005

HttpUnit, Client Side Testing Framework for Web applications, <http://httpunit.sourceforge.net/>, last accessed, August 14, 2005

IEEE (2002). :IEEE standard glossary of software engineering terminology. Standard 610.12-1990 (R2002), IEEE Computer Society Press.

IEEEMultimedia, <http://ieeexplore.ieee.org/xpl/>, last accessed, September 14, 2005

J2EE, Java 2 Enterprise Edition (J2EE), Online: <http://www.jcp.org/en/jsr/detail?id=168>, Accessed: July 14,2005, <http://java.sun.com/j2ee>, last accessed,

JCP, Java Community Process, <http://jcp.org>, last accessed, August 8, 2005

Jetspeed, Enterprise Jetspeed Portal, <http://portals.apache.org/jetspeed-2/>, last accessed, August 20, 2005

JMeter, Performance Testing of Web applications using Apache JMeter, <http://jakarta.apache.org/jmeter/>, last accessed, August 20, 2005

JSF, JavaServer Faces Technology, <http://java.sun.com/j2ee/javaserverfaces/>, last accessed, September 14, 2005

JSR168, Java Specification Request (JSR) 168 Portlet API Specification Version 1.0, <http://www.jcp.org/en/jsr/detail?id=168>, last accessed, July 14, 2005

JUnitPerf, Performance and Scalability testing using JUnitPerf, <http://www.clarkware.com/software/JUnitPerf.html>, last accessed, 2005, August 14

jWebUnit, Client Side Testing of Web applications using jWebUnit, <http://jwebunit.sourceforge.net/>, last accessed, August 14, 2005

Kaner, C., Bach, J. and Pettichord, B. (2002). **Lessons Learned in Software Testing**, John Wiley & Sons Inc.

Kaplan, B., and Maxwell, J. A (1994). Qualitative Research Methods for Evaluating Computer Information Systems, Sage, Thousand Oaks, CA: 45-68.

Kastel, B. (2003). **Enterprise Portals For the Business and IT Professional**, *Competitive Edge International*.

Kitchenham, B. and Pfleeger, S. L. (2002). "**Principles of survey research: part 5: populations and samples.**" *SIGSOFT Softw. Eng. Notes* **27**(0163-5948): 17-20.

Krutchén, P. (1999). **The Rational Unified Process**, Addison Wesley.

Kung, D. C., Liu, C.-H. and Hsia, P. (2000). An Object-Oriented Web Test Model for Testing Web Applications 24th International Computer Software and Applications Conference IEEE Computer Society: 537-542

Lee, A., Resolution: What is the place of Surveys?, <http://www.ucalgary.ca/~newsted/ppt/sld021.htm>, last accessed,

Loveland, S., Miller, G., Jr, R. P. and Shannon, M. (2004). **Software Testing Techniques**, Charles River Media.

Lucca, G. A. D., Casazza, G., Penta, M. D. and Antoniol, G. (2001). An Approach for Reverse Engineering of Web-Based Applications Proceedings of the Eighth Working Conference on Reverse Engineering (WCRE'01) IEEE Computer Society: 231

Lucca, G. A. D., Fasolino, A. R. and Tramontana, P. (2004). "**Reverse engineering web applications: the WARE approach** " *J. Softw. Maint. Evol.* **16** (1-2): 71-101

Lucca, G. D., Fasolino, A. and Faralli, F. (2002). Testing Web Applications Proceedings of the International Conference on Software Maintenance (ICSM'02) IEEE Computer Society: 310

Mackinnon, T., Freeman, S. and Craig, P. (2000). **Endo-Testing: Unit Testing with Mock Objects**. *Extreme Programming and Flexible Processes in Software Engineering (XP2000)*.

Mackinnon, T., Freeman, S. and Craig, P. (2001). Endo-testing: unit testing with mock objects Extreme programming examined Addison-Wesley Longman Publishing Co., Inc.: 287-301

Massol, V. and Husted, T. (2003). **JUnit In Action**, *Prentice Hall, Upper Saddle River, NJ*.

Maven, Apache Maven Project, <http://maven.apache.org/>, last accessed, August 14, 2005

Mercury and LoadRunner, Performance and Load Testing using Mercury LoadRunner, <http://www.mercury.com/us/products/performance-center/loadrunner/>, last accessed, August, 2005

MockObjects, Testing with Mock Objects, <http://c2.com/cgi/wiki?MockObject>, last accessed, August 12, 2005

Nguyen, H. Q., Johnson, B. and Hackett, M. (2003). **Testing Application on the Web**, *Wiley Publishing Inc.*

Oracle, Oracle Portal Server, <http://www.oracle.com/technology/products/ias/portal/pdk.html>Oracle, last accessed, August 20, 2005

Patton, M. Q. (2002). *Qualitative Research & Evaluation Methods*, Sage Publications California.

Pawlak, Z. (1992). *Rough Sets: Theoretical Aspects of Reasoning about Data*, Kluwer Academic Publishers.

Pfleeger, S. L. and Kitchenham, B. A. (2001). "**Principles of survey research: part 1: turning lemons into lemonade** " *SIGSOFT Softw. Eng. Notes* **26** (6): 16-18

Pipka, J. U. (2002). *Test- Driven Web Application Development in Java*, Daedalos Consulting GmbH.

Pluto, Portlet Container Reference Implementation for Java Portlet Specification, <http://portals.apache.org/pluto/multiproject/pluto/>, last accessed, August 14, 2005

PortalZone, DeveloperWorks Websphere Portal Zone, <http://www.ibm.com/developerworks/websphere/zones/portal/>, last accessed, August 14, 2005

PortletCommunity, <http://community.java.net/portlet/>, last accessed, September 14, 2005

PortletUnit, Java Unit Testing Framework for testing JSR-168 portlets, <http://sourceforge.net/projects/portletunit/>, last accessed, August 14, 2005

Sandbox, Sandbox Systems, Online :: <http://sandportal.sandboxsystems.com/wps/portal>
Accessed: August, 2005, last accessed,

Schulmeyer, Gordon, G. and Mackenzie, G. R. (2000). **Verification and Validation of Modern Software-Intensive Systems**, *Upper Saddle River, NJ: Prentice Hall*.

ServletUnit, Testing Servlets using ServletUnit, <http://servletunit.sourceforge.net/>, last accessed, August 14, 2005

Shilakes, C. and Tylman, J. (1998). Enterprise Information Portals, Industry Overview. Proceedings of the International Conference on Software Maintenance (ICSM'02), Merrill Lynch, New York: 310.

Smith, S. and Meszaros, G. (2001). **Increasing the Effectiveness of Automated Testing**. *XP Universe*, Raleigh, NC.

Struts, Portlet Struts Framework, <http://publib.boulder.ibm.com/infocenter/wpdoc/>, last accessed, September 1, 2005

TDD and Unit-Testing, TDD and Unit Testing, Online : <http://www.testdriven.com>
Accessed: August, 2005, last accessed,

UserEngineering, <http://www-306.ibm.com/ibm/easy/>, last accessed, September 14, 2005

Walsham, G. (1995). "Interpretive Case Studies in IS Research: Nature and Method." *European Journal of Information Systems*: 74-81.

WebspherePortal, WebSphere Portal (V5.02) InfoCenter, <http://publib.boulder.ibm.com/pvc/wp/502/index.html>, last accessed, July 14, 2005

Wege, C. and Chrysler, D. (2002). Portal Server Technology June 2002. IEEE Internet Computing: 73-77.

Wenliang Xiong, Harpreet Bajwa and Maurer, F. (Jul 2005). **WIT: A Framework for In-container Testing of Web-Portal Applications**. *Web Engineering: 5th International Conference, ICWE*, Sydney, Australia, Springer-Verlag.

WIT, Web Portlet In container testing framework, Online: <http://godzilla.cpsc.ucalgary.ca/ebe/Wiki.jsp?page=Root.Portlettesting>, Accessed: September 15,2005, last accessed,

Wosnick, S., Developing and Unit Testing with Cactus (IBM WebSphere Developer Technical Journal), http://www-128.ibm.com/developerworks/websphere/techjournal/0206_wosnick/wosnick.html, last accessed, August 14, 2005

WSRP, O. (2003). Web Service for Remote Portlets Specification Version 1.0, September 2003. Online: <http://www.oasis-open.org/committees/download.php/3343/oasis-200304-wsrp-specification-1.0.pdf> Accessed: July 14,2005.

Yin., R. K. (2003). **Case Study Research, Design and Methods**, *Sage Publications: Thousand Oaks, California*.

Appendix A. Portal Technology

A.1 Review of Portal Server Framework Services

Product Name	Organization	Description of the services provided	JSR 168
Pluto	Apache	<p>Pluto is the reference implementation for JSR168, the Java portlet specification. Pluto serves as a portlet container that implements the Portlet API and offers developers a working example platform from which they can test their portlets. The project comes with a minimal portal for testing.</p> <p>http://portals.apache.org/pluto/</p>	Yes
Jetspeed 2	Apache Jakarta	<p>Jetspeed is an Open Source implementation of the Enterprise Information Portal, using Java and XML. Jetspeed-2 is in Beta version and is conformant to the Java Portlet Standard. Jetspeed provides support for content publication frameworks.</p> <p>http://portals.apache.org/jetspeed-2/</p>	Yes
UPortal version 2.2	JA-SIG	<p>uPortal is an open source portal under development by institutions of higher-education with the development effort shared among several of JA-SIG member institutions. Presented as a set of Java classes and XML/XSL documents, it provides a framework for producing a campus portal.</p> <p>http://www.uportal.org/</p>	Yes
EXo platform	EXo Platform SARL	<p>The eXo platform software is an Open Source corporate portal and content management system. The components include a portlet container which is a certified implementation of JSR168. The enterprise version comes with its own application server and workflow management tools. The product comes with a content</p>	

		management system and a services container. http://www.exoplatform.org/portal/faces/public/exo	Yes
GridSphere	GridLab Project	GridSphere portal framework provides an open-source portlet based Web portal. It enables developers to quickly develop and package third-party portlet web applications that can be run and administered within the GridSphere portlet container. http://www.gridsphere.org/gridsphere/gridsphere	Yes
Websphere Portal & Portal Toolkit	IBM	The IBM Portal Toolkit, Version 5.0.2.2/5.0.2.3 provides the capabilities to customize, create, test, debug, and deploy individual portlets and Web content. IBM also has a proprietary API within WebSphere Portal. However, with JSR 168 standards, it is recommended that portlet developers use the new standardized portlet API. The foundation of the platform is IBM WebSphere Application Server. For portlet development needs, Portal Toolkit plugs into the IBM WebSphere Studio development environment (WSAD). http://www.ibm.com/developerworks/websphere/zones/portal/	Yes
WebLogic Portal	BEA	It provides an enterprise portal platform for production and management of custom fit portals. Provides portlet wizards for the creation of different portlets (JSP/HTML, JSR168, Struts, WSRP) http://dev2dev.bea.com/products/wlportal81/index.jsp	Yes
Sun Java System portal Server 6	Sun	A Java portal that works with a number of application servers, providing additional development tools and utilities, single sign-on for aggregated applications to the portal, supports the creation and consumption of Web services-based portlets and incorporates the J2EE platform. http://www.sun.com/software/products/portal_srvr/home_portal6.xml	Yes
Oracle		Application Server Portal provides a framework for integrating content from external sources. A number of	

AS Portal	Oracle	<p>different components are available, and the Portal Development Kit (PDK) comes with a Portlet Container for building and running interoperable Java portlets. An extension for JDeveloper provides a wizard for the step-by-step creation of portlets. The PDK enables developers to build portlets in any web accessible language including Java/J2EE, Web Services, ASP, PL/SQL, XML</p> <p>http://www.oracle.com/technology/products/ias/portal/pdk.htmlOracle</p>	Yes
Vignette V7 Portal Services	Vignette	<p>Vignette comes as an application portal and a builder for creation, assembly and customisation of applications. Pre-defined portlets are available with the portal, including one for integrating .NET Web applications as portlets. The builder is intended to support portlet development</p> <p>http://www.vignette.com/</p>	Yes

A.2 Portlet API

This section explains briefly the basic interfaces, methods and core objects of the Portlet API referenced in the thesis.

Abstract portlet class: is the central abstraction of the Portlet API. All portlets extend this abstract class by extending one of its subclasses for example PortletAdapter.

PortletRequest: interface represents the user's request and encapsulates information about the user and the client. An implementation of the PortletRequest is passed to the delegated do methods (doView, doEdit etc).

PortletResponse: interface encapsulates the response sent to the Portal Server for aggregation.

PortletSession: object is used to store information needed between requests. This object is created when a user logs into the portal.

PortletContext: provides a mechanism for the portlet to access the services of the portlet container in which it is running.

PortletData: object represents a portlet instance on a user page and contains user specific data.

PortletSettings: object encapsulates the configuration information of the portlet instance. In addition this object can be used as storage for attributes to be shared by all portlet instances.

Service Methods: The portal calls the service methods when the portlet is required to render its content. These methods are doView, doEdit or doHelp.

Appendix B. Testing Tool Evaluation

Testing Techniques	Tools	Limitation & Applicability with respect to Portal applications	Tool Description
Unit Testing	jUnit	Cannot test methods that need environment context; Portlets, servlets & JSPs cannot be tested	JUnit is standard unit testing framework for testing java classes
Unit testing Using Mock Objects (out of container)	ServletUnit	Supports testing of servlet logic . Cannot ensure servlets will execute correctly in the real servlet container	Extends JUnit by executing unit tests in a simulated servlet container; Access to servlet request, response and session objects to set up specific request state for testing is provided
	PortletUnit	Supports fine grained testing of JSR 168 portlets. Cannot ensure portlets will execute correctly in the real portlet container	Extends JUnit and evolved from ServletUnit and Pluto container. Access to portlet request, response and session objects is provided to set up specific request state for portlet testing
Integration Unit testing (in container)	Cactus	Cactus does not support Portlet testing.	Extends JUnit; supports in container testing of servlets, EJBs, filters. Each test case has 3 methods begin, test and end. Begin method runs in simulated client environment, provides data & creates request and response objects; second method executes on server by calling server side code & uses context provided by begin method. In end method response is verified.

Functional Unit testing	httpUnit	Portal Applications can be tested. Limitations are 1) Setting up the initial test state is time consuming 2)html element identifiers are dynamically generated using portlet namespace encoding 3)Parsing html for response validation extremely slow	Tools extend JUnit and provides a means for simulating a browser & querying web server, Externally to verify response.
	HtmlUnit	-	Provides a testing approach similar to httpUnit. Difference is manner in which the response is returned document versus response object
	jWebUnit	-	Java testing framework that combines functionality of HttpUnit with JUnit simplifying the creation of test cases
	Canoo	-	Provides a testing approach similar to httpUnit. Test case can be specified as use cases. Test execution supported using ANT for easy integration into an existing ANT build.

Appendix C. Survey Materials

C.1 Survey Introduction Form

December 15th, 2004.

Hello Portal Developers,

I am a research student at the e-business Engineering group at the University of Calgary. I would like to invite members of the Web-portal development community to participate in our research study designed to:

- Evaluate the current process of developing and testing web-portal applications.
- Understand the testing practices in use and challenges (if any) in testing portal applications.
- Provide us with feedback that will assist in making recommendations and potential improvements in the development and testing process of web-portal applications.

Participation is voluntary. If any questions disrespect your privacy please feel free to decline to answer them. To maintain confidentiality findings shall be reported as an aggregate. The data shall be summarized to draw conclusions and no individual participant can be identified from the results. No individual names shall be used in academic presentations and publications. The results of the study will be shared with all the participants.

Your willingness to complete this survey will help immensely in building the body of knowledge of web-portal application development and testing processes. For more information on this project visit: <http://www.cpsc.ucalgary.ca/~bajwa/research.html>

Please return the completed survey to Ms.Harpreet Bajwa by e-mailing at bajwa@cpsc.ucalgary.ca at the earliest convenience.

Thank you in advance for your participation.

Sincerely,
Harpreet Bajwa

C.2 Survey Questionnaire

Part 1: BACKGROUND

1. a) How many years of experience do you have building:

Web Applications _____ years

Web-Portal Applications _____ years

b) Describe all the roles you have worked as part of software development?

c) Specify the role you play at present in web-portal application development

2. What kind of software development process does your organization use at present for developing web-portal applications (e.g. XP, other agile methods, RUP, RAD etc)

3. In general, describe the nature of web portal applications you build (e.g. educational, enterprise, news portals etc)?

4. Which Portal framework do you currently use to:

Develop and Test portal applications (i.e. test and development environment)

Deploy Portal application (i.e. production environment) _____

5. Does your organization have:

QA Team

Software Testing Team

PART 2: PORTAL APPLICATION DEVELOPMENT AND TESTING

1. For each of the following practices, specify what kind of techniques you use for testing web-portal applications?

(Please answer the associated questions with the choices, you select)

Unit Testing

a. Which framework is used to write the unit tests?

b. How are the unit tests run i.e. automated test scripts, manual?

c. Who writes the unit tests?

d. Who runs the unit tests?

e. How often are the unit tests run within your team?

Frequently Sometimes almost never

Functional/ Black box testing

a. Which framework is used to write the functional tests?

b. How are the functional tests run i.e. automated test scripts, manual?

c. Who writes the functional tests?

d. Who runs the functional tests?

e. When are the functional tests run?

*Server side testing.

a. Which framework is used to write the server side tests?

b. How are the tests run i.e. automated test scripts, manual

c. Who writes the server side tests?

d. Who runs the server side tests?

e. When are the server side tests run?

Performance/Scalability testing

a. Which framework is used to write the tests?

b. How are the tests run i.e. automated test scripts, manual

c. Who writes the tests?

d. Who runs the tests?

e. When are the performance/scalability tests run?

User acceptance testing

Specify other testing techniques used.

2. Have you ever experienced errors that show up when staging the web- portal application from the development to the production portal server?

YES

NO

If you selected yes above,

a) Describe the errors, their severity briefly and how the errors are fixed?

b) Are any tests written for the production portal server environment? Please explain.

3. Describe the challenges in testing web-portal applications and areas where automated test

support might be beneficial?

Based on the responses a selected group will be invited for an interview. Please indicate your preference

Would like to be invited

Would NOT like to be invited


Please provide your telephone numbers here_____.

A follow up e-mail shall be sent to confirm an interview date and time.

For any other question or concerns please e-mail me at bajwa@cpsec.ucalgary.ca

*By server side testing we imply testing of methods such as doView, doEdit etc that require context from the environment they are running in. In case of portlets we are referring to the portlet container.

Appendix D. Ethics Approval



**UNIVERSITY OF
CALGARY**

CERTIFICATION OF INSTITUTIONAL ETHICS REVIEW


This is to certify that the Conjoint Faculties Research Ethics Board at the University of Calgary has examined the following research proposal and found the proposed research involving human subjects to be in accordance with University of Calgary Guidelines and the Tri-Council Policy Statement on "Ethical Conduct in Research Using Human Subjects". This form and accompanying letter constitute the Certification of Institutional Ethics Review.

File no: **4111**
 Applicant(s): **Harpreet Bajwa**
Wenliang Xiong
 Department: **Computer Science**
 Project Title: **Evaluating, Predicting and Improving the Process of Testing Web Applications (Portal) Using Server-Side Testing Tool Support. Investigating the Performance of Server-Side Testing Tool Provided to Test Web (Portal) Applications and Assess its Usefulness and Ease of Use**
 Sponsor (if applicable):

Restrictions:

This Certification is subject to the following conditions:

1. Approval is granted only for the project and purposes described in the application.
2. Any modifications to the authorized protocol must be submitted to the Chair, Conjoint Faculties Research Ethics Board for approval.
3. A progress report must be submitted 12 months from the date of this Certification, and should provide the expected completion date for the project.
4. Written notification must be sent to the Board when the project is complete or terminated.


Janice Dickin, Ph.D., LL.B.
Chair
Conjoint Faculties Research Ethics Board

2004/09/24
Date:

Distribution: (1) Applicant, (2) Supervisor (if applicable), (3) Chair, Department/Faculty Research Ethics Committee, (4) Sponsor, (5) Conjoint Faculties Research Ethics Board (6) Research Services.

2500 University Drive N.W., Calgary, Alberta, Canada T2N 1N4 • www.ucalgary.ca

Appendix E. Perceptive Study Materials

E.1 Pilot Study at U of C

With Students and followed by Sandbox employees

In-Container Testing Using WIT Framework

Overall Instructions: We will be writing and running a very simple in-container test using the WIT framework for the doView method. This test will make sure the data generated by the portlet is available and correct. We have already provided in your workspace 1) PortalProject called HelloWorld with a single HelloWorld Portlet and 2) The WITInvokerClient that acts a testClient invoking the tests and gathering the test results.

Writing a TestCase for the HelloWorld Portlet

1. Write a test class called [HelloWorldPortletTest](#) that extends testCase to test the doView method of the HelloWorld portlet. Use the example test Class provided on your desktop as a guide for writing the test.

Note: InvokerWIT folder under the root path: [lib\WITLibs](#) must contain the following jars as part of the WIT install:

- aspectjrt, aspectjtools, commons-httpclient-3.0-alpha2, jdom, xerces, **PortletTester**.

[These jars have been included in the build path of the portlet project under test.](#)

Running the TestCase for the HelloWorld Portlet

Step 1: Inspect the InvokerWIT folder. You should see the following configuration files

which you will modify as follows:

1. **build-wps properties:** Modify the downloaded build-WIT properties in the places marked in the file. Alternatively make sure you have the following attributes changed in the file.

portlet_src_files_dir_path: the root path of portlet source code.
portlet_tst_src_files_dir_path: points to the root path of the test code.
bin_path: points to the root path of the portlet binary class files.

AntControllerServlet:

<http://localhost:9081/HelloWorld/AntControllerServlet>

2. **WIT-Config.xml:** Modify the WIT-Config.xml by making changes at marked locations in the file.
3. **WIT-Config.xsd**
4. **build-wps.xml**

Step2: Change the web.xml file of the portal project under test: Add the servlet [AntControllerServlet](#) with the servlet mapping to the URL pattern.

Step3: Run the build file build-wps.xml: You can run the build file from the command prompt > ant -f build-wps.xml

Alternatively you can do step wise run by:

1. run ant -f build-wps.xml testCase2Aspect
2. run ant -f build-wps.xml weaving
3. run ant -f build-wps.xml deploy
4. run ant -f build-wps.xml start2Test

You should be able to see the testResults!!!!

Note: If you see exceptions showing httpURL connection while running the build it maybe due to the following:

- 1) Check to ensure AntControllerServlet URL in the build-wps properties is correct with the correct contextroot.

-contextRoot of the portlet Application can be changed using the .websettings file under your project.

-change the context Root in corresponding EAR file too.

- 2) Check web.xml of the portal application under test to make sure the AntControllerServlet is added.
- 3) Check to make sure the PortletTester.jar is in the build path of the project under test.
- 4) Make sure step1, step2 are followed correctly and values have been changed to point to the correct locations.

E.2 Consent Forms

Research Project Title:

Evaluating, predicting and improving the process of testing web applications ([portal](#)) using the testing tool support.

Investigator(s): Harpreet Bajwa, Wenliang Xiong, Frank Maurer (bajwa, xiong, maurer)@cpsc.ucalgary.ca

This consent form, a copy of which has been given to you, is only part of the process of informed consent. It should give you the basic idea of what the research is about and what your participation will involve. If you would like more detail about something mentioned here, or information not included here, you should feel free to ask. Please take the time to read this carefully and to understand any accompanying information.

Description of Research:

The purpose of this research is to identify an improved process for testing web portal applications by conducting empirical studies and as a result improve the quality of web applications. The researchers also aim to assess the perceived benefits of the automated testing approaches developed and determine from its 1) Perceived Ease of use i.e. can it be used with less effort) 2) Perceived Usefulness i.e. will using the tool result in any advantage which will determine its future usage.

Procedure:

By checking on “**I agree to participate in the study**”, you will be a part of the study consisting of the two phases:

1) As part of the first phase a presentation will be given that describes the testing approaches together with writing and running testcases using the testing framework. 2) As part of this phase, the developers will record their perceptions of the presented testing approaches through a questionnaire provided by us followed by interactive discussions and follow up interviews. Some interview sessions shall be recorded to facilitate collection of information and transcribed for analysis with your permission. You will be sent a copy of the transcript and results to confirm accuracy and clarify any points.

Likelihood of Discomforts:

There is no harm, discomfort, or risk associated with your participation in this research.

Confidentiality:

Participant anonymity will be strictly maintained. No information that discloses your

identity will be released or published without your specific consent to disclosure. All the data collected will be stored in a password-protected computer and only be accessible to the investigators.

Primary Researcher(s):

Harpreet Bajwa and Wenliang Xiong are M.Sc. students in the Department of Computer Science at the University of Calgary under the supervision of Dr. Frank Maurer. This study is conducted as part of their research and will be included in their thesis.

By checking on the **“I agree to participate in the study”** checkbox, you indicate that you have understood to your satisfaction the information regarding participation in the research project and agree to participate as a subject. In no way does this waive your legal rights nor release the investigators, sponsors, or involved institutions from their legal and professional responsibilities. You are free to withdraw from the study at any time. Your continued participation should be as informed as your initial consent, so you should feel free to ask for clarification or new information throughout your participation. If you have questions concerning matters related to this research, please contact any of the three investigators of this research.

If you have any questions or issues concerning this project that are not related to the specifics of the research, you may also contact the Research Services Office at 220-3782 and ask for Mrs. Patricia Evans.

I agree to participate in the research study

Participant's Signature

Date

Harpreet Bajwa

Investigator and/or Delegate's Signature

Date

Lawrence Liu

Witness' Signature

Date

A copy of this consent form will be given to you to keep for your records and reference.

E.3 Post Study Questionnaire

1. How **likely** are you to use WIT in **your company environment** in the future to improve the testing process?
 Very Unlikely Unlikely Quite likely likely Very likely
 Please provide us with reasons for your choice

2. Could you highlight any other specific areas where portal application testing can be made more **effective** and **comprehensive**?

3. Which of the following **do you perceive** is the ICT testing approach using WIT helpful in:
 - Detecting/diagnosing Portlet Deployment Related Errors
 Not at all Very little Average Helpful Very helpful

 - Problems arising from the Interaction between the container & Application Code
 Not at all Very little Average Helpful Very helpful

 - Security Role Based Resource Access
 Not at all Very little Average Helpful Very helpful

 - Others, please specify

4. In your opinion at what stage of **portal application development process** is writing and running ICT tests likely to be beneficial? (i.e. When/Where would such tests be run?)

5. Who is likely to **write & run the ICT tests** in your company environment?

6. You perceive that **Writing** ICT tests using WIT is:

Very Hard Hard Average Easy Very Easy

Please provide us with reasons (if any) for your choice

7. You perceive that **Running** ICT tests using WIT is:
 Very Hard Hard Average Easy Very Easy

Please provide us with reasons (if any) for your choice

8. How **likely** are you to use **Portlet Unit testing using Mock Objects (MO)** in **your company environment** to develop and test Portlets
 Very Unlikely Unlikely Quite likely likely Very likely

How do you **perceive** to incorporate MO into the **existing testing Process**?

Any other comments?

We would like to invite you for an interview. Please indicate your preference

- Would like to be invited
 Would NOT like to be invited

If you are willing to participate in an interview, please provide your telephone numbers and/or email here:

WIT is an open Source Project and can be downloaded from:
<http://godzilla.cpsc.ucalgary.ca/ebe/Wiki.jsp?page=Root.WITUsage>

Appendix F. Co-Author Permission



November 7th, 2005

University of Calgary

2500 University Drive NW

Calgary, Alberta

T2N 1N4

I, Frank Maurer give Harpreet Bajwa permission to use co-authored work from our papers “Evaluating Current Testing Processes of Web-Portal Applications” and “WIT: A Framework for In-container Testing of Web-Portal Applications,” for Chapters 3, 4 and 5 of her thesis.

Sincerely,

A handwritten signature in black ink, appearing to be "F. Maurer", written over a horizontal line.

Frank Maurer



November 7th, 2005

University of Calgary

2500 University Drive NW

Calgary, Alberta

T2N 1N4

I, Wenliang Xiong give Harpreet Bajwa permission to use co-authored work from our papers “Evaluating Current Testing Processes of Web-Portal Applications” and “WIT: A Framework for In-container Testing of Web-Portal Applications,” for Chapters 3, 4 and 5 of her thesis.

Sincerely,

A handwritten signature in black ink that reads "Wenliang Xiong" followed by the date "Nov 8, 05".

Wenliang Xiong
Nov 8, 05

Wenliang Xiong

Appendix G. Materials and Raw Data

1. Survey materials and Questionnaires are in SurveyMaterials Folder
2. Interview transcripts and notes are located in the SurveyTelephoneInterviews Folder
3. Sandbox discussion notes and transcripts of the discussions are located in the Case Study Folder
4. Charts and Raw data are included in the Survey.xls excel file.
5. Presentation slides and materials for portal developers on March 18 (pilot study) and August are included in the Empirical Analysis folder
6. Empirical study Questionnaire is in the Empirical Analysis folder
7. Charts and Raw data are included in the Empirical Analysis folder Analysis.xls excel file.
8. Application Source Code and tests referred in Section 5.2.2 is included in ShippingDemo Folder
9. Rough Set analysis files are in the Rough Set analysis folder in files called surveydata.isf and rules.isf

Glossary

Application Server is a term used to describe a set of components that extend their services to other components.

Build are versions or redesigns of a project that is in development. A given project may undergo numerous revisions before it is released

Container is the interface between a component and the low-level platform-specific functionality that supports the component.

Database Servers act as data repositories for Web applications. Most web based systems use relational database servers.

Enterprise JavaBeans (EJB) container manages the execution of enterprise beans for J2EE applications.

JVM, Java virtual machine, JVM is a platform-independent execution environment that converts Java bytecode into machine language and executes it.

J2EE is a distributed application-server environment that provides a set of java extension APIs to build enterprise applications.

Java Server Pages (JSP) provides for the generation of dynamic content for the Web.

Modules are parts, components, units or areas that comprise a given project. Modules can also be thought of as units of software code.

Mock - a mock object adds to this the ability to preload an object with data which can be played back during a test. Most importantly, it

is also given details of the expected interactions that other objects will have with it and at the end of testing verify that these have taken place.

Portlet Preferences provides portlet developers a mechanism for persisting configuration, customization and personalization settings for individual users. Portlet preference interface provides methods for writing, reading and resetting preferences. They are made available to all PortletRequests.

Enterprise portals (or “corporate desktops”) give employees access to organization-specific information and applications.

Enterprise Java Bean (EJB) is a collection of Java classes and an XML file bundled into a single unit. EJB runs in an EJB container. They form the business logic components deployed in a J2EE environment. Also associated with the persistence layer.

Regression Testing is a form of testing that makes sure that a program has not regressed which means verifying that the functionality that was working yesterday is still working today.

Role is a set of permissions. Users are mapped to roles. The administrator who has the authority to manage a portal page can grant access to view, edit, delete, portlet resources.

Servlet implements a request-response paradigm especially for Web environment and run in the servlet container

Stub object is the most minimal implementation of a class providing nothing more than the implementation of an interface.

Test Plan is a document that defines the inputs and expected outputs, format of the output, expected execution time and testing environment. It also outlines risks, priorities and schedules for testing.

Test Case is a test that executes a single well defined test objective (e.g. a specific behaviour of a feature under a specific condition)

Test Suite is a collection of logically related test cases

Test Scripts are step by step instructions that describe how a test case is executed.

Test Types are categories of tests that are designed to expose a certain class of errors.

Web or Http Servers store web pages or html files and their associated components and make them available to web based clients.

Web-container manages the execution of JSP and servlet components.